Mohammad Rokonuzzaman Pappu

Online Model Predictive Control of a Robotic System by Combining Simulation and Optimization

School of Electrical Engineering
Department of Electrical Engineering and Automation

Thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Technology

Espoo, August 18, 2015

Instructor: Professor Perttu Hämäläinen
Aalto University
School of Arts, Design and Architecture

Supervisors: Professor Ville Kyrki
Aalto University
School of Electrical Engineering

Professor Reza Emami
Luleå University of Technology

Preface

First of all, I would like to express my sincere gratitude to my supervisor Professor Ville Kyrki for his generous and patient support during the work of this thesis. He was always available for questions and it would not have been possible to finish the work in time without his excellent guidance. I would also like to thank my instructor Perttu Hämäläinen for his support which was invaluable for this thesis. My sincere thanks to all the members of Intelligent Robotics group who were nothing but helpful throughout this work. Finally, a special thanks to my colleagues of SpaceMaster program in Helsinki for their constant support and encouragement.

Espoo, August 18, 2015

Mohammad Rokonuzzaman Pappu

Aalto University
School of Electrical Engineering Abstract of the Master's Thesis

Author:	Mohammad Rokonuzzaman Pappu
Title of the thesis:	Online Model Predictive Control of a Robotic System by Combining Simulation and Optimization
Date:	August 18, 2015 **Number of pages:** 11+70
Department:	Automation and Systems Technology
Programme:	Master's Degree Programme in Space Science and Technology
Professorship:	Automation Technology (AS-84)
Supervisors:	Professor Ville Kyrki (Aalto)
	Professor Reza Emami (LTU)
Instructor:	Professor Perttu Hämäläinen

In the field of robotics, model predictive control is considered as a promising control strategy due to its inherent ability to handle nonlinear systems with multi-dimensional state spaces and constraints. However, in practice, the implementation of model predictive control for a nonlinear system is not easy, because it is difficult to form an accurate mathematical model for a complex nonlinear system. Moreover, the time required for solving a nonlinear optimization problem depends on the complexity of the system and may not be suitable for real-time implementation.
In this thesis, a general approach for implementing model predictive control for nonlinear systems is proposed, where a physics-based simulator is used for the prediction of the states and a stochastic optimization based on particle belief propagation is used to solve the optimization problem. To study the ability of the controller, a nonlinear robotic system is built.
The designed controller is capable of handling nonlinear system for both single variable and multiple variables. For the current system, the controller is unable to solve the optimization problem in real time with the presence of constraints.
The proposed method provides a simpler approach for implementing model predictive control, which can be used for a range of robotic applications. However, in this method, the capability of the controller depends on the physics engine's ability to simulate different physical systems and the speed and accuracy of the physics engine.

Keywords: Model Predictive Control, Physics Engine, Particle Belief Propagation.

Contents

List of Tables

List of Figures

List of Algorithms

Abbreviations

CFM	Constraint Force Mixing
ERP	Error Reduction Parameter
FPS	Frame Per Second
FRI	Fast Research Interface
GPC	Generalized Predictive Control
LMPC	Linear Model Predictive Control
LWR	Light Weight Robot
MPC	Model Predictive Control
MRF	Markov Random Field
NMPC	Nonlinear Model Predictive Control
ODE	Open Dynamic Engine
OROCOS	Open Robot Control Software
PBP	Particle Belief Propagation
ROS	Robot Operation System
UDP	User Datagram Protocol

Symbols

α	State potential
β	Input potential
γ	Potential function of vector $\mathbf{Z}$
Γ	Set of neighboring nodes of a node
$\psi(\cdot)$	Potential function of a node
$\Psi(\cdot)$	Potential function of a pair of nodes
b	Belief at a node
c	Clique of a graphical model
$\mathbf{C}$	Diagonal covariance matrix
G	Graphical model
$\mathbf{i}$	Cost of input
i_b	Sample index of the best trajectory
$\mathbf{J}(\cdot)$	Cost function
m	Message from one node to another
N	Number of samples
N_{th}	Effective sample number
$\mathbf{P}(\cdot)$	Probability density function
q	Proposal distribution
$\mathbf{s}$	Cost of state
u	Input variable
$\mathbf{u}$	Input vector
$\mathcal{U}$	State space of u
V	Set of nodes in a graphical model
x	State variable
$\mathcal{X}$	State space of x
$\mathbf{x}$	State vector
w	Process noise
$\hat{y}$	Estimated output
$\mathbf{Z}$	Vector containing state and input variables

Chapter 1

Introduction

Robotic hardware is getting more capable and complex at an enormous rate with the improvement of science and technology, along with the persistent work of a lot of researchers and engineers. However, with the increased complexity, it is also getting increasingly difficult to design efficient controllers for the robots.

To design an efficient controller for a robotic system, simulation has been an important tool over the last few decades [1]. Simulation is the process of building a realistic model of the system and observing the operation and the outcome of the system by executing the model. With the increasing speed and capability of modern computers, simulators are also getting increasingly powerful and efficient, which makes it possible to simulate more complex systems with better accuracy.

Optimization is the process of choosing the best element from an available set of elements using specific criteria. Generally, in control engineering problems, a performance index or cost function is formed based on the outcome of the system, which then needs to be maximized or minimized depending on the desired goal. Optimization is considered to be one of the most important tool for designing and operation of any system, as a properly optimized system can have much higher efficiency [2].

However, the problem with simulation has always been the fact that it does not provide an optimized result. Simulation can be carried out for a range of reasonable inputs and the best system configuration can be chosen based on the corresponding outputs. Unfortunately, this does not guarantee the best possible solution and a sub-optimal result might get chosen during this process. To overcome this problem, researchers were interested in combining simulation with optimization, which is stated in literature as "simulation optimization" or "optimization via simulation" [3]. By combining simulation and optimization, it is possible to find the best possible solution, without explicitly evaluating all

possible input parameters. The motivation behind combining simulation and optimization is to use minimum resources to obtain maximum information from a simulation model.

Model predictive control (MPC) is a framework for online control, which requires an explicit model of the system to generate control actions [4]. The controller uses the model to predict the future possible outcomes of the system. A performance index or cost function for the predicted states is formulated based on the outcomes and the goal of the system. Finally, an optimized input (the best possible outcome) is generated based on the predicted states and corresponding cost functions using an optimizer.

The purpose of this thesis is to control a robotic system in real time by combining simulation and optimization using a model predictive control approach. In this thesis, simulation is used as a replacement of the mathematical model for predicting the future states of the system. The simulation generates future outcomes of the system and an optimizer is used to find the best possible solution from the predicted states. Moreover, for optimization a stochastic approach is used based on the simulation. In the stochastic optimization framework, the process includes one or more parameters that are random in nature. Stochastic optimization approaches are capable of handling highly nonlinear, high dimensional systems which are inappropriate for classical deterministic optimization algorithms [5]. For this work, a generic sample based belief propagation technique called "particle belief propagation" [6] is used, which is a message-passing algorithm that can be used to solve inference problems, such as finding the marginal distribution of random variables using a graphical model [6].

1.1 Motivation

In robotics, model predictive control has been a topic of interest due to the fundamental nature of different robotic systems, such as the presence of multiple variables, nonlinearity and constraints. Model predictive control has significant advantages over conventional controllers for nonlinear multivariable systems [7]. It provides inherent robustness for handling nonlinear systems, with constraints on both the state and control variables. Moreover, due to the intuitive nature of MPC, tuning is much easier and provides intrinsic compensation for dead time [4].

The most important aspects of MPC which separates it from other control techniques are following: 1. It needs an accurate model of the system. Future outcomes of the system are predicted using the model. The model should be accurate enough to predict outcomes close to the real system. 2. It solves an optimization problem at each time interval based on the current and previous

state of the system. Only the first control action will be provided to the real system as the whole process is repeated for the next time step.

However, the same properties that make it different from other control techniques, pose some difficulties in its implementation. The main difficulties are the formation of an accurate model for a higher dimension nonlinear system and the computational power required for optimization after each time step for a complex system.

Even though it is possible to design an accurate model of a low dimensional system, it is extremely difficult to form a detailed and accurate mathematical model of an inherently nonlinear, multivariable, higher order system. Moreover, MPC performs optimization after each time step, which makes it computationally expensive. With the rise of the complexity of the system model, designing the model become more challenging and also the optimization problem requires significantly increased computational burden, which may not be suitable for real-time implementation. Due to this problem, the use of MPC in the 80's and 90's was rather limited to industrial processes, where the system is slow enough to perform the optimization after every sample [7].

Meanwhile, due to the enormous increase in the computation speed of CPUs, the interest is growing to use MPC for the systems with faster dynamics. In the last decade, substantial research been done to improve the theoretical and practical aspects of MPC. To avoid the complexity of nonlinear models for implementing MPC, researchers have used simplified linearized models of the system [8–12]. However, linearization of nonlinear system produces a few difficulties. The most important one is the introduction of uncertainty due to the model mismatch, which affects the performance of the controller and in the worst case scenario makes the system unstable. Researchers have also tried to use the nonlinear model and subsequently solve the nonlinear optimization problem [13–15]. However, the complexity of nonlinear optimization problem depends on the system model, which is much more computationally expensive and might be prohibitive for the real-time implementation for complex systems, even for modern computers.

Simulation has been a frequently used tool for complex systems, where it is impossible or at least difficult to find an optimized solution analytically [16]. With the improvement of physics based simulators, it is now possible to model and simulate complex robotic systems in the simulator with reasonable accuracy [17]. Physics based simulators, generally called as physics engines, enable the user to simulate any physical system. They incorporate the law of physics while managing interaction between different objects and model the motion of rigid bodies in a physical world.

Considering the improvement of physics based simulators with the increase in computational speed, prediction of the future states required for a model predictive controller can be achieved by using a physics based simulator. This will

remove the need of forming a complex mathematical model for each and every system. Moreover, currently available physics engines are capable of simulating rigid bodies faster than real time [17, 18], which also provides the opportunity to use simulation for online optimization.

Stochastic algorithms for optimization have been used for robust control design as an alternative of deterministic numerical optimization [19, 20]. The deterministic optimization depends on the complexity of the model which makes it challenging for implementing in a nonlinear system. On the contrary, for a stochastic algorithm the complexity of optimization does not depend on the system model [21]. It obtains the optimal solution by sampling the set of potential solutions a number of times, from where the best sample is chosen based on their performances. This simplifies the analysis and design tasks, which makes it suitable for high-dimensional nonlinear systems. Implementations of stochastic MPC have already been proposed in [21–23].

Recently, Erez et al. [24] proposed a design for a real-time implementation of MPC, based on a physics engine. Their final goal was to control a humanoid robot with a combination of automatic control and human interaction. In their approach, a human operator provides the high-level guidance to the robot and a model predictive controller is used for low-level automatic control. The implemented controller used a physics engine for the prediction of the states. Similarly, Kumar et al. [25] proposed a model predictive control approach for the manipulation of dexterous robotic hand. In their work, they also used a physics engine for the prediction of the future states.

On the other hand, Hämäläinen et al. [26] implemented a MPC algorithm for online motion synthesis of a 3D human character based on a physics engine. Another physics engine based MPC was proposed by Hämäläinen et al. in [18]. In both cases, authors used a stochastic algorithm for the optimization. In [26] a Sequential Monte Carlo sampling was used, while in [18] the authors used a Particle belief propagation based optimization technique. The purpose of both works was to generate adaptive and reactive responses for 3D characters in any unknown environment.

Unfortunately, all the work stated above is still limited to simulation, any implementation in real robotic hardware is yet to be reported. However, with the successful implementation of online MPC for complex systems in the simulator, we believe that it is possible to implement MPC based on a physics engine for a real robotic system.

1.2 Objective

The thesis has two objectives. The first objective is to propose a simple, efficient and more general solution for the implementation of MPC in robotic systems by combining simulation and optimization. The use of a physics engine for the prediction of the future states provides a more generalized approach for modeling the system dynamics in MPC context. On the other hand, due to the use of stochastic optimization technique based on the simulation, the complexity of optimization is no longer dependent on the system model and can be used for more general applications. The second objective is to build a robotic system with nonlinear system dynamics, to test the functionality of the designed controller, where the final goal is to test the capability of the controller to handle a nonlinear system with multiple variables and constraints.

1.3 Outline

The thesis outline is as follows. Following the introduction, a general overview of model predictive control is given in Chapter 2. This chapter also includes the previous implementations of model predictive control in the field of robotics. Chapter 3 includes the background theory for the particle belief propagation technique. In Chapter 4, a general overview of the design of the model predictive control by combining simulation and optimization is presented. On the other hand, in Chapter 5, detail of the specific implementation of the proposed MPC algorithm for a robotic system is discussed. Chapter 6 includes the details of the experiments carried out to test the capability of the designed controller. The results of the experiments are also discussed in this chapter. Finally, Chapter 7 includes the overall conclusion of the thesis work.

Chapter 2

Model Predictive Control

Model predictive control, in general, does not refer to a specific control technique. It is a framework for online control, including a range of control algorithms, that uses dynamic model of the system to generate control actions. In this framework, the controller predicts the future states of the system using the model and solves an optimization problem at each time interval over a finite horizon. The optimized control is applied to the system and based one the feedback from the process the optimization process is repeated over the shifted horizon.

The main difference between MPC and other conventional control techniques is the manner by which it is being implemented. The conventional controller uses off-line feedback policy, whereas, the MPC solves an online optimization problem. However, the biggest advantage of MPC is its ability to handle systems with multidimensional state space and constraints. Constraints play an important role in the controller design of any robotic system. Different constraints of the system need to be considered while designing a controller for robotic systems. For example, to design a suitable path planner for mobile robots, different constraints, such as geometric, dynamic and kinematic constraints needs to be taken into account. Evidently, MPC provides excellent performances while handling complex multivariable system with constraints in the process industries [27, 28], which is why it is popular in the process industries. Moreover, the MPC control strategy depends on solving an optimization problem based on a cost function or a performance index. These cost functions or performance indexes can be formulated intuitively using constraint on both state and control input, which provides the opportunity to use it for a range of applications.

2.1 Formulation and strategy

In this section, working principle of a model predictive controller is described. Moreover, an example of formulating a typical model predictive controller for a nonlinear system is presented.

MPC strategy : A model predictive controller is comprised of three distinct elements, common for all algorithms used in MPC context: prediction model objective/cost function and optimization. Different methods can be used to implement these elements, which results in different algorithms for implementing model predictive control. Figure 2.1 shows the basic structure of a model predictive controller.

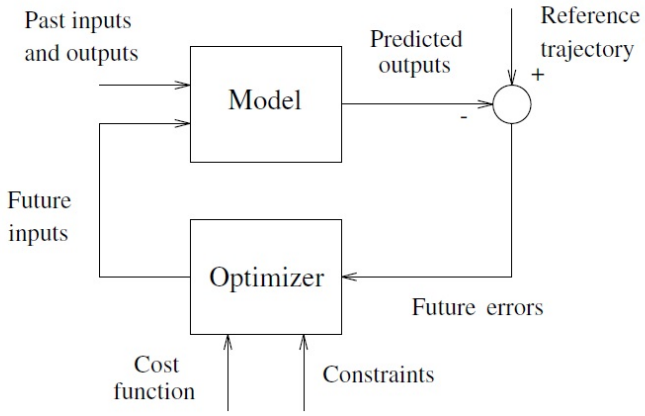


Figure 2.1: Basic structure of MPC [4].

In general, a mathematical model of the system is required which predicts the output of the system. It is possible to formulate the model of the system using different strategies, for example, impulse response model, step response model, transfer function model and state space model. Even though, in research literature the state space model has always been preferred [29], in process industries impulse response model is widely used [4].

At each time step t, the future states are predicted using the process model for a fixed horizon N_p, called a prediction horizon. Future outputs $\hat{y}(t+k|t)$, where $k = 1....N_p$, are estimated for a range of future control inputs $u(t+k|t)$, where $k = 0.....N_p - 1$. The prediction of the states are based on the previous and current inputs and outputs of the process model until time t.

The cost function is a way of assigning a value to a certain action of the system. Using this cost function, it is possible to assign a number to a system response, based on the corresponding input, state variables and output. Generally, a cost function indicates how close the system is to the desired outcome. The optimization process uses the cost function to generate control actions until

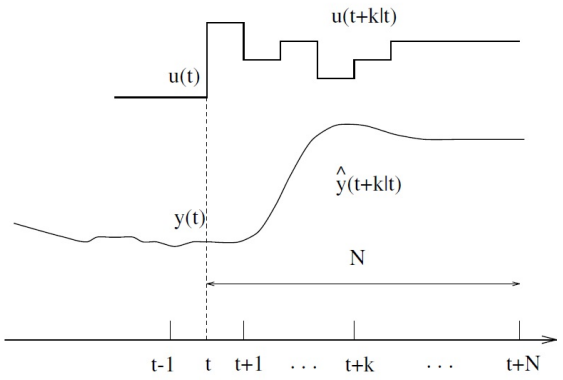


Figure 2.2: MPC strategy [4].

a certain prediction horizon. Generally, the purpose of the optimization is to maximize or minimize the cost function. Optimization techniques may differ depending upon the cost functions and the complexity of the system.

The optimization process yields an optimized control sequence $u(t+k|t)$. However, only the control $u(t|t)$ is sent to the system and following controls are discarded. This whole process is repeated for the next time step.

MPC Formulation: As we are interested in designing a model predictive controller for a nonlinear system, a typical formulation of MPC for a nonlinear system is described below.

Assuming state variables ($x \in \mathbb{R}^n$) and inputs ($u \in \mathbb{R}^m$), a discrete time nonlinear system can be expressed as

$$x(t+1) = f(x(t), u(t)), \tag{2.1}$$

with the following constraints

$$x(t) \in \mathcal{X}, \ \forall t \geq 0, \tag{2.2}$$
$$u(t) \in \mathcal{U}, \ \forall t \geq 0, \tag{2.3}$$

where $x(t)$ is the state and $u(t)$ is the input variables, $\mathcal{U}$ is the set of feasible input values and $\mathcal{X}$ is the set of feasible state values. For the nonlinear system described above the optimization problem can be formed as

$$\underset{\mathbf{x}(t),\ \mathbf{u}(t)}{min} \ \mathbf{J}(\mathbf{x}(t), \mathbf{u}(t)), \tag{2.4}$$

assuming,

$$x(t+k|t) \in \mathcal{X}, \quad u(t+k|t) \in \mathcal{U}, \tag{2.5}$$
$$x(t+k+1) = f(x(t+k|t), u(t+k|t)), \quad k = 0, ...N_p, \tag{2.6}$$

where N_p is the prediction horizon, $x(t+k|t)$, $u(t+k|t)$ are the predicted values of $x(t+k)$, $u(t+k|t)$ respectively at time t and $\mathbf{x}$, $\mathbf{u}$ are the vectors that contain the predicted states and input sequences.

$$\mathbf{x}(t) = [x(t+1|t)^T .....x(t+N_p|t)^T]^T \tag{2.7}$$

$$\mathbf{u}(t) = [x(t|t)^T .....x(t+N_p-1|t)^T]^T \tag{2.8}$$

The cost function for the system can be formulated as

$$\mathbf{J}(\mathbf{x}(t), \mathbf{u}(t)) = \sum_{i=0}^{N_p-1} x(t+k|t)^T P x(t+k|t) + u(t+k|t)^T Q u(t+k|t) + x(t+N_p|t) R x(t+N_p|t), \tag{2.9}$$

where P, Q and R are the weighting matrices. To solve the optimization problem different techniques can be used [30]. Assuming a solution of the optimization problem is available, the first step of the optimized sequence is sent to the system and the same process is repeated for the next time step. Algorithm 1 shows the general algorithm for model predictive control.

Algorithm 1 General MPC algorithm.

1: **Input**: Initial state, previous inputs and outputs, state and control constraints.
2: **Output**: Optimal control.
3: Form the optimization problem (2.4)-(2.9).
4: **while** MPC is running **do**
5: Measure current state $x(t)$.
6: Solve the optimization problem.
7: Apply only $u(t|t)$.
8: t = t+1.
9: **end while**

2.2 Model predictive control in robotics

The use of model predictive control has been proposed for a variety of robotic applications. This section presents a brief discussion of previous works on model prediction control in the field of robotics.

Wheeled mobile robots: Model predictive control has always been an interesting prospect for wheeled mobile robots due to the fact that humans use

predictive control to drive a car. Moreover, the formulation of the system model and the solution of the optimization problem is much easier for wheeled mobile robot because of the low dimensionality of the system.

Generally, the control of a mobile robot requires following tasks: perception, path planning, path tracking and low-level control [8]. However, MPC has generally been proposed for the path tracking task in mobile robot control. The use of MPC is promising for path tracking because MPC provides excellent performance when the reference trajectory is known [9]. For the path tracking task, reference trajectory is known and also an accurate kinematic model of the system can be formed, which makes it profoundly suitable for MPC.

One of the early implementations of MPC for the wheeled mobile robot was by Ollero [8], where an implementation of Generalized predictive control (GPC) was presented based on the work proposed by Clarke [31]. The Authors used a linearized model of the system, which does not always provide desirable performance. Other implementations of MPC using linearized model can be found in [9–11].

Linearization of a nonlinear system is not always desirable because linearization introduces uncertainties in the system. Moreover, if the nonlinear system operates over a large dynamic range, the simplification is not practically viable anymore and requires the use of a nonlinear model predictive control (NMPC) [32].

Even though NMPC is a well-established theory in the control regime [33, 34], it poses some difficulties in its practical implementation. In NMPC, a solution of nonlinear optimization problem needs to be solved online, which is computationally expensive, most cases impossible for online optimization. On the other hand, due the nonlinear nature of the system, it requires to solve a convex for optimization, a difficult problem to be solved in practice [32].

Meanwhile, *Neural Network* is proved to be a useful tool to form functions of nonlinear systems, a few researchers proposed neural network approach for MPC [13, 35, 36]. Camacho [13] proposed a neural network based approach for mobile robot path tracking for a nonlinear system.The advantage of using a neural network based approach is its ability to model a complex nonlinear system [35]. This approach also reduced computational cost for sensor data processing, which supposedly made the real-time implementation of MPC possible.

However, the use of neural network for mobile robots has also faced some well known problems of neural network implementation. The main drawbacks of using neural network were the presence of undesirable local minima and the slow nature of the convergence of back-propagation learning [35]. Moreover, the optimization method used in [13] is no longer efficient when the system have high dimensionality and used for long ranges. The efficiency of the neural network depends on the learning speed and learning effectiveness, which does

not always meet the criteria for online optimization and consequently use of MPC [37].

Robotic manipulators: Due to its wide range of applications, modeling and controlling of robotic manipulator was one of the most important research fields over the last 20 years in the field of robotics. Any articulated robot consists of two or more joints can be considered as a complex nonlinear dynamic system. However, in most of the industrial applications, the control problem was simplified as linear [38].

Boscariol et al. [12] proposed a MPC controller for a four-link flexible mechanism using simulation. A linearized dynamic model was used for the flexible manipulator. In the simulation, it provided better performance than the conventional PID controller. However, the effect of uncertainty was not considered and as shown in the result that performances degrade with the addition of noise in the model.

As a flexible link manipulator is highly nonlinear in nature and linearization affects the performance of the controller, a lot researchers were compelled to use a neural network based approach, as the neural network provides excellent performance when a nonlinear system is considered. A handful of researchers used nonlinear MPC based on the neural network in [14, 15, 38, 39].

A comparative study between GPC and a neural network based GPC was done by Durmus et al. [40]. They used the Lagrange-Euler approach for creating the model of the system. GPC used a linearized model of the system for optimization and a nonlinear model was used for neural-network based implementation. In this study, it was also proved that due to the use of linearized version of the system model, the performance of the controller degrades significantly. According to their simulation, the motion of the manipulation is more smooth and flexible in case of the Neural network based MPC than the general GPC.

Humanoid Robots: Due to the inability of the wheeled mobile robots to move in rough terrains and mimic human behaviors, humanoid robot has been at the center of interest in the field of robotics for the last two decades. Effort has been made in the field of humanoid robot to emulate the human behavior as closely as possible, which makes the use MPC an important prospect due to fact that human uses predictive approach for their movement such as walking, running, jumping and balancing.

Dimitrov et al. [41] proposed a linear MPC for walking pattern generation of a bided robot based on zero moment point model. In this approach, a linearized model of the system with constraints was used. A problem of quadratic program was solved for the optimization. Even though there were several algorithms to solve a quadratic program problem, the "Active set" approach was used by the authors. Same authors also suggested in [42] that with this approach it is possible to generate continuous walking gait online. Implementation of linear

MPC for humanoid robots can also be found in [43, 44], where any reference trajectory was unknown. The target for both the work was to mimic the human walking behavior as closely as possible because human does not always follow a specified trajectory to reach a goal but uses optimization technique based on changing environment, trend to fall, obstacles etc.

Recently, Henze et al. [45] proposed another model predictive control approach for humanoid robots, where it requires multiple contacts to complete a task. A control strategy for the whole body is implemented, where the job of the controller is to ensure posture stabilizing and balancing. Piperakis et al. [46] presented a successful implementation of model predictive control on a practical humanoid robot. A Nao humanoid robot was used for the implementation. A constrained LMPC approximated by an orthonormal basis was used.

Aerial robots: A feasibility study for the implementation of MPC for a rotorcraft based unmanned vehicle was done by Kim [47]. A nonlinear MPC was used for the study, which was implemented for an autonomous helicopter.

Shim et al. [48] presented a study which showed the possibility of controlling multiple autonomous aerial vehicle using nonlinear MPC in a complex environment. Keviczky [49] presented a comparative study between a linear and nonlinear MPC based on its probable use in an autonomous aerial vehicle. A f-16 aircraft model was used to carry out the study. They provided an excellent result about the computation time needed in both the cases. According to the authors, even though the linear MPC provides satisfactory results when computation time was concerned, the nonlinear MPC was yet not feasible for with current computational speed at 2006. However, the simulation was done in real time using a Pentium III single core computer, the possibility of its implementation with faster computer today is yet to be studied.

But for the linearized MPC, the linearization of the model was not general. A flight conditioned linearization was required for the proper implementation of the controller, which is cumbersome to perform linearization for every specific condition.

Other: MPC was also proposed for other kinds of robots than the ones described above. Ginhoux et al. [50] and Bebek et al. [51] proposed the use of model based technique for robotic assisted beating heart surgery. And recently Dominici [52] proposed another MPC based approach for robotic assisted beating heart surgery. The purpose of their work was to cancel the relative motion between the beating heart and the surgical instrument. However, Dominici [52] improved the approach by implementing a Kalman active observer, which provided significantly better result. Even though it provided better results, it is not yet perfectly suitable for use for the complex and complicated task of surgery.

Summary: Based on the literature, it is evident that the most important bottlenecks for implementing online MPC are the complexity of forming the

system model and the required computational speed for online optimization. It is possible to implement linear MPC for nonlinear systems, but it does not always provide desirable results. The robustness and stability of linear MPC are not yet proven for the highly nonlinear system, due to the uncertainty introduced by linearization of the model. Moreover, if the system operates over a large dynamic range, use of a linearized model is no longer a viable option.

On the other hand, implementing nonlinear MPC is not always viable, a lot of cases it is prohibitive for online implementation, due to the computational burden required for a higher order fast dynamic system. To solve this problem, researchers used few other techniques among which neural network provided significantly improved performance. However, the formation of the neural network problem is complicated and labor intensive.

Even though linear MPC could be used for lower order systems, the linearization of any model is operation specific. To carry out linearization for different operations is labor intensive and not usable in more general context. Due to the above-mentioned reasons, it is necessary to look for an alternative solution.

Chapter 3

Particle Belief Propagation

For the optimization in the current work, a sampling based belief propagation method is used. This chapter describes the background theory for the technique, called particle belief propagation.

3.1 Markov random field

Probabilistic graphical models have proved to be powerful tools to represent and visualize dependencies between a large number of random variables [53]. They are a combination of graph theory and probability theory, which makes it easier to form large-scale probabilistic models. In other words, they combine probability and logical structure, which provides a simple framework for modeling and solving optimization and inference problem [54]. Moreover, probabilistic graphical models can be used to approximate quantities, such as means or marginal distributions, which are essential for message-passing algorithms like belief propagation [6]. The two most commonly used graphical models are a directed graphical model or a Bayesian network and a undirected graphical model or a Markov random field.

A Markov random field (MRF) represents conditional dependencies between random variables. Like any graphical model, a Markov random field consists of nodes and edges. Every node corresponds to a random variable and edges represent the probabilistic interaction between the connected pair of nodes. However, in a Markov random field the direction of the interaction between random variables are unknown. This is useful especially when it is required to model systems, where the directionality of interaction between the variables is not known [54]. Figure 3.1 shows an example of a Markov random field, where nodes are connected to each other by undirectional edges.

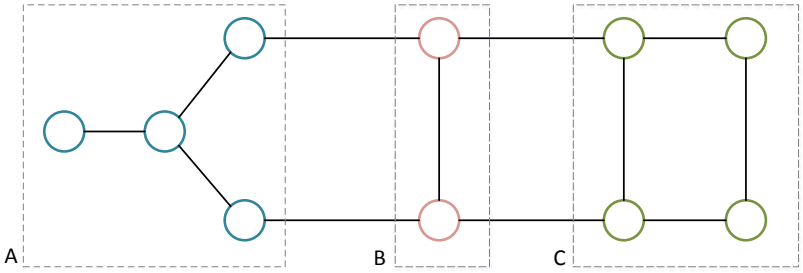


Figure 3.1: An example of a Markov random field, where each set A, B and C contains a number of nodes.

The modeling of a Markov random field is based on the conditional dependencies between the random variables. A random variable x_A is said to be conditionally independent of x_B, given the observation of the variable x_C, if

$$P(x_A|x_B, x_C) = P(x_A|x_C). \tag{3.1}$$

In other words, if the probability of x_A is not affected by the observation of x_B, it can be said that x_A is independent of x_B. However, this dependency is based on the observation of x_C. This dependency relationship between a large number of random variables can be represented intuitively using the graphical models. In the graphical model shown in Figure 3.1, any node in set A is conditionally independent of any node in C, given the set B, due to fact that all the paths between nodes in A and C are blocked by the nodes in B.

To build a mathematical formulation of Markov random field, let us assume a graphical model $G = (V, E)$, where V is the set of nodes and E is the set of undirectional edges. For each node k ($k \in V$), x_k is the corresponding random variable, χ_k is the state space of x_k ($x_k \in \chi_k$), $\mathbf{x}$ is the joint random variable $(x_k)_{k \in V}$ and Γ_k = set of neighbors of node k.

A process can be stated as a Markov random field, if the edges E of the graph G are undirected and satisfies the following local Markov property that *every variable is conditionally independent of the remaining variables given its neighbors.* In this context, the graphical model G can be stated as a Markov random field, if

$$\forall k \in V,\ x_k \perp x_{v-k} | x_{\Gamma_k}, \tag{3.2}$$

which states that in a Markov random field, x_k is conditionally independent of other random variables (x_{v-k}), given the neighbors of corresponding node k.

Let us consider the graphical model G as a simple first order Markov chain, shown in Figure 3.2. In this network, node 1 has one neighbor, which is node 2. Similarly, node 3 also has one neighbor, node 2. Using the local Markov property

it can be said that node 1 is independent of node 3 given its only neighbor node 2 and vice-versa, which is equivalent to our previous description of conditional dependency, where it was stated that if the paths between two nodes are blocked by observed third node, the nodes are conditionally independent.

In the network shown in Figure 3.2, variable x_1 is conditionally independent of x_3 because the path between x_1 and x_3 is blocked by node k = 2. Using this consideration the joint probability of the network P($\mathbf{x}$) can be written as

$$P(\mathbf{x}) = P(x_2)P(x_1|x_2)P(x_3|x_2). \tag{3.3}$$

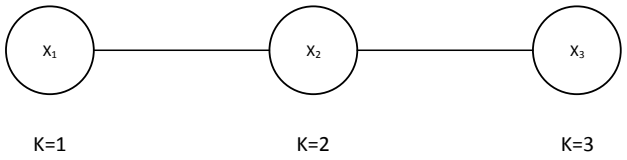


Figure 3.2: A simple Markov random field.

However, joint probability in a Markov random field can also be expressed in terms of potential functions of the nodes, using the relationship between a Markov random field and a Gibbs random field. The Hammerly-Clifford theorem, which unifies the Markov random field and the Gibbs random field, can be used to form a more generalized function to calculate joint probabilities of random variables in a Markov random field.

Similar to the Markov random field, the Gibbs random field also uses graphical models to represent a set of random variables and the relationship between the random variables. In a Gibbs random field, the joint probabilities are expressed using the function of cliques. A clique is a subset of an undirected graph which is complete [55]. In other words, A clique is a fully connected subset of the graph. According to the Gibbs theory, the joint probability of any set of random variables can be expressed as the product of the potential function of each clique. If we assume, C is the set that contains all the cliques c_k of a Gibbs field, then the joint probability of the set of random variables $\mathbf{x} = x_k$, where $k = 0, 1....n$ is

$$P(\mathbf{x}) = \frac{1}{M} \prod_{c_k \in C} \psi(c_k), \tag{3.4}$$

where $\psi(c_k)$ is the potential function of the clique c_k and M is a normalization constant.

Evidently, both Markov random field and Gibbs random field uses an undirected graph to represent the relationship between the random variables. According to the Hammerly-Clifford theorem, in a same graph Markov random field and Gibbs field are equivalent to each other. In the graphical model of Figure 3.2, the joint distribution P($\mathbf{x}$) over the discrete random vector $\mathbf{x}$ is a Gibbs distribution, if and only if the random vector $\mathbf{x}$ make up a Markov random field with respect to the graph G. To summarize, in a Markov random field the probability distribution can be factorized over the clique of the graph.

For simplicity and relevance to the work, we consider a pairwise Markov random field approach. In a pairwise Markov random field, the cliques consist of either one node or a pair of nodes, where the pairs are the nodes of the same edge [56]. Incidentally, this setting suggests that the pairwise Markov random field has two types of potential function, the potential function of the node and the potential function of the edge (pair of nodes connected to the edge). Finally, using the Hammerly-Clifford theorem, the probability distribution for the pairwise Markov random field can be defined as

$$P(\mathbf{x}) = \frac{1}{M} \prod_{k \in V} \psi_k(x_k) \prod_{k,s \in E} \Psi_{k,s}(x_k, x_s), \tag{3.5}$$

where ψ_k is the potential function of variable node k and $\Psi_{k,s}$ is the potential function for the edge that connects node k and node s. Equation 3.5 represents a general form to calculate joint probabilities in a Markov random field because the pairwise Markov random field model is very general and any graphical model can be transformed into a pairwise Markov random field [57].

3.2 Belief propagation

Belief propagation is a message-passing algorithm which can be used to solve inference problems, such as finding the marginal distribution of random variables in a graphical model. Belief propagation is a parallel iterative form of dynamic programming, where it performs a dynamic programming recursion when the system under consideration is a simple tree or chain [58]. Due to this, it provides correct solutions for optimization and inference problems.

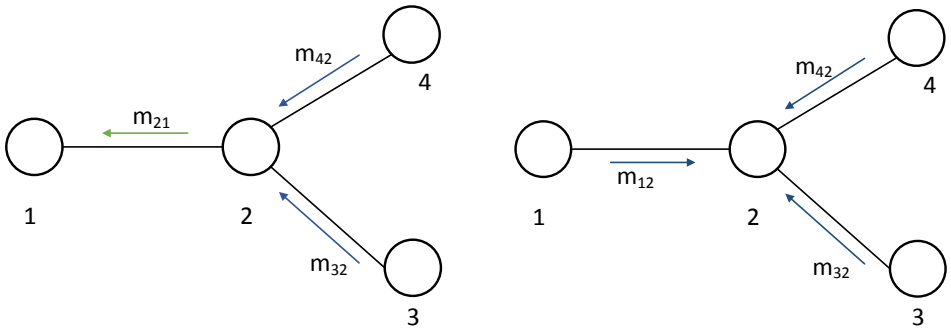
In belief propagation, neighboring nodes pass messages to each other and after enough iteration, the messages converge. At each iteration, each node receives messages from each neighboring nodes and sends messages to its neighbors. Each node updates its message based on the message received from the neighboring nodes in each iteration. After the convergence based on the received messages, the belief for each node is calculated.

For a pairwise Markov random field, joint probabilities can be described using (3.5). Let us assume in a graphical model, x_s and x_k are two neighboring nodes.

A message from node s to k, $m_{s:k}$, contains the information about the state of target node k, according to source node s. The message essentially informs that according to s, how likely it is for k to be in the corresponding state. A message from the node s to node k can be written as

$$m_{s:k}(x_k) = \sum_{x_s} \Psi_{k,s}(x_k, x_s)\psi_k(x_s) \prod_{s \in \Gamma_s / k} m_{s:k}(x_s), \tag{3.6}$$

where Γ_s is the set of neighboring nodes of s. The message from node s to node k is sent recursively, which contains the message from all the other nodes connected to s, except k. Figure 3.3 shows an example of message passing in Markov random field. Figure 3.3(a) shows that node 2 send the message to node 1 after collecting the messages from all its neighboring nodes except node 1. In the next step, shown in Figure 3.3(b), it collects the messages from all the other nodes (including node 1) and updates its message.



(a) (b)

Figure 3.3: (a) Message from node 2 to node 1. (b) Node 2 collects messages to update its message.

Equation (3.6) represents the message that passes from one node to another. This message passing continues until the messages converges, After convergence, the belief at any node t can be calculated as

$$b_k(x_k) = \psi_k(x_k) \sum_{s \in \Gamma_k} m_{s:k}(x_k). \tag{3.7}$$

Moreover, if the graphical model is a simple tree, belief $b(x_k)$ of any node k, represents the marginal distribution of node k [53, 59].

3.3 Particle belief propagation

In a Markov random field, it is possible to solve an inference problem for parameters such as marginal distribution of a random variable using belief propagation.

If the Markov random field is a simple tree general belief propagation can be used, otherwise, loopy belief propagation technique needs to be used [60]. These approaches work well when the state space of each variable is relatively small or the parametric form of the distribution is Gaussian. However, for continuous-valued variables the effective state spaces need to be reduced, which can be accomplished by either discretizing the state space or by random sampling [53]. Within the random sampling based approaches, a particle filter is the most popular technique. The particle belief propagation is a generalization of the particle filter, which can be used for more general graphical models [53].

In particle belief propagation, for each node k, a set of particles $x_k^1....x_k^N$ are generated from a proposal distribution $q_k(x_k)$, where $q_k(x_k) > 0$. The belief of any node k can be calculated as [6]

$$b_k(x_k^i) = \psi_k(x_k^i) \prod_{\Gamma_k} m_{s:k}(x_k^i), \tag{3.8}$$

and message from node s to node k can be written as

$$m_{s:k}(x_k^i) = \frac{1}{N} \sum_{j=1}^{N} \psi(x_s^j) \Psi(x_s^j, x_k^i) \prod_{s \in \Gamma_s / k} m_{ks}(x_s^j), \tag{3.9}$$

where i and j are sample index and N is the number of samples.

The optimization problem in this thesis involves solving an inference problem, where the marginal probability of the states are calculated using particle belief propagation. To find an optimized trajectory, a number of trajectories are predicted from where the best trajectory is chosen. Each trajectory segment is produced by generating samples at each time interval. Particle belief propagation is used to generate good trajectories, where each trajectory segments connects with the next one to form a full trajectory until the prediction horizon.

Chapter 4

MPC Combining Simulation and Optimization

This chapter provides a general overview of the designing aspect of a model predictive controller by combining simulation and optimization. The developed algorithm for the proposed model predictive control is discussed in detail.

4.1 Simulation

Simulation plays an important part in the design of the proposed model predictive controller, as the efficiency of the controller depends on the accuracy of the simulation. Physics engines are simulators, which enable the user to simulate any physical system. They incorporate the laws of physics while managing interaction between different objects and model the motion of the bodies in a physical world. However, the dynamics of the bodies handled by physics engines separate them into two different categories, soft body dynamics and rigid body dynamics. The simulator that incorporates soft body dynamics provides the opportunity to simulate deformable objects, meaning that objects under consideration may deform during simulation. Some simulators handle only rigid bodies, where it is assumed that object under consideration does not deform.

In this thesis, we require a simulator, which handles rigid body dynamics as there are no deformable objects used for the experiments. Generally, a rigid body dynamic simulator has two separate phases of simulation, collision detection and dynamic response. In collision detection phase, the simulator takes account all the objects and finds the objects that will collide with each other. After finding the objects which will collide with each other, it calculates the

contact points for the objects. In the next stage, based on the detected collision points, the corresponding response of the system is calculated, which is then used to simulate the system forward. The detail of the working principle of a physics engine is outside the scope of this thesis, however, interested readers can find the working principle in [61, 62].

Physics engines are extensively used in video games, animations and dynamic simulations [63, 64]. There are a few open source physics simulators available among which Open Dynamic Engine (ODE), Bullet, Havok and PhysX are the most popular [17]. However, the choice of a physics engine for robotic applications is not straightforward. None of the physics engines was built for a specific discipline, which is why each physics engine has their specific strengths and weaknesses. Hummel et al. [65] presented a comparative study between ODE, Bullet, PhysX, Havok and Newton. The authors used the following five criteria for comparing the physics engines: collision computation, collision response, constraint stability, interpenetration and advance collision and friction. According to their study, Havok was the fastest engine with the expense of accuracy. On the other hand, Newton and PhysX were stable under constraints, but their computation time for collision detection was significantly higher than the others. Bullet and ODE perform similar for all the tests, however, ODE performed reasonably fast and accurate when the number of collisions was not too large. Other similar works can be found in [66–68], however, the most recent and relevant work for this thesis can be found in [17], where the authors compared the physics engines based on their performances for different applications including different robotic systems. Moreover, this comparison includes a new physics engine "MuJoCo" created by the authors [69]. The authors used four different models for simulation which are a 35-DOF robotics arm grasping a capsule, 25-DOF humanoid model, 5-DOF planar kinematic chain and stack of capsules. According to their work, it was again suggested that no physics engine outperforms others in all the aspects. According to this work, the new physics engine MuJoco was the fastest and the most accurate for robotic applications such as grasping and humanoid robots. On the other hand, ODE performed best for simulating disconnected bodies (stack of objects). Overall, ODE remained reasonably accurate, even though it was significantly slower than MuJoCo. MuJoCo would have been the best option for this work due its combined speed and accuracy for robotic applications, but unfortunately it was not available as an open source at the time this thesis work started, which is why ODE was chosen.

ODE is a constraint-based physics engine that uses Euler integration for simulation [70, 71]. ODE has two main components, rigid Body dynamics and collision detection Engine. In ODE, any mechanical system can be represented by bodies and the joints. A joint connects the bodies and defines the rotation and orientation between the bodies. ODE has several joint types using which it is possible to simulate a real mechanical system. For example, a simple robotic arm with 3 links can be constructed in ODE using two joints shown in Figure 4.1, where body 1 corresponds the base link.

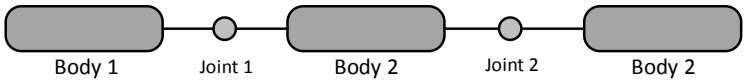


Figure 4.1: Simple robotic arm with 3 links in ODE.

Like any physics simulator, ODE requires following inputs for its operation: description of the system (model), system state, force/torque to control the system and time step. Based on these parameters the simulator generates the system state after the provided time step. An overview is shown in Figure 4.2.

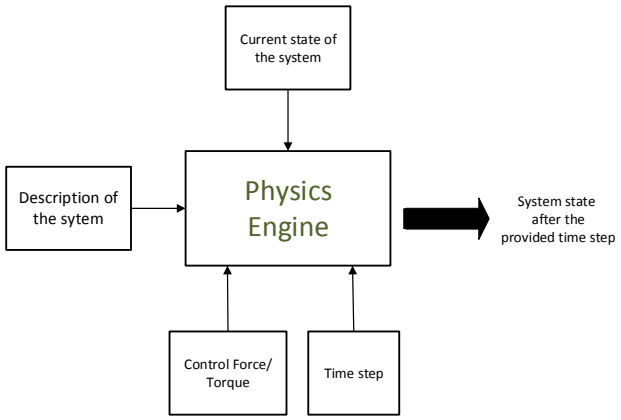


Figure 4.2: Operation with ODE.

4.2 MPC combining simulation and optimization

We discuss a general approach toward designing a path finding algorithm in MPC context, where the target is to find an optimized trajectory for a robotic system. The specific implementation of this algorithm will be presented in Chapter 5. MPC provides the opportunity to use any generic model for the prediction of the states. In this case, a simulator is used to predict the future outcome of the system. Moreover, a novel optimization technique based on particle belief propagation is used. The optimization algorithm is adopted from [18], where it was used for motion synthesis of 3D character animation in a simulator.

Figure 4.3 shows the general structure of the MPC by combining simulation and

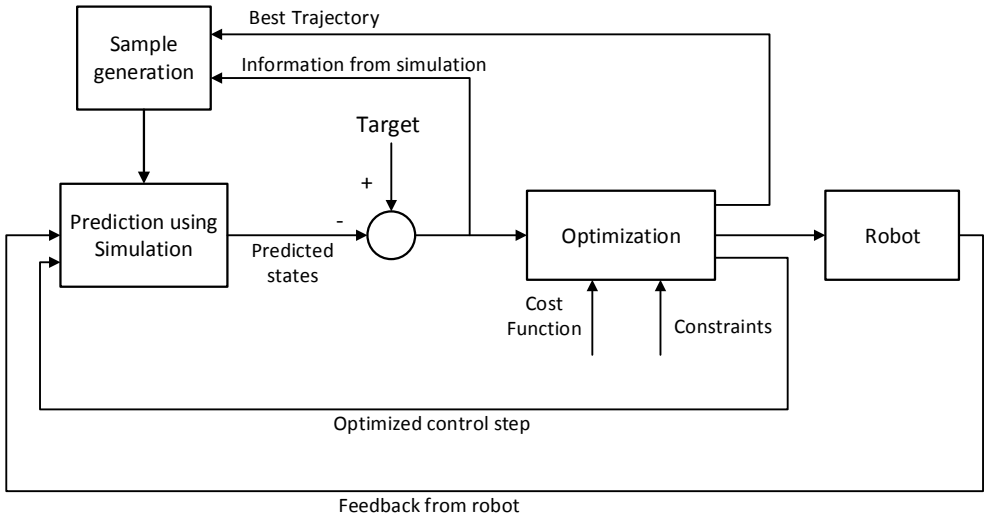


Figure 4.3: Structure of the proposed model predictive controller.

optimization. The sample generator generates a number of control samples from a proposal distribution, which is modified based on the marginal distributions of the samples of the previous step. Marginal distributions of the samples are calculated using particle belief propagation technique. Control samples are then used by the simulator to generate predicted outputs of the system.

Samples were generated at each time step, which produce trajectory segments for each sample using simulation. Particle belief propagation technique was used to ensure that each trajectory segment connects with the next segment, which produces a full trajectory until the prediction horizon. This essentially produces a number of trajectories equal to samples of each control step. Every trajectory is associated with a cost and based on the cost optimizer chooses the best trajectory. The first step of the chosen trajectory is sent to the real robot as a control signal and information of the best trajectory is used to generate samples for the next control step by the sample generator. The first step of the best trajectory is also sent to the simulator, which is used to move the simulation forward. Moreover, a feedback about the system state is collected at each time step, which is used to correct the possible error between the simulation and the real system. The detail of the approach is described next.

As it was discussed the previous section, the ODE physics engine is used for the prediction of the states. ODE is a black box dynamics simulator, where only the current and future state variables are known based on any given input. The simulator is used to simulate the system forward until the prediction horizon. In this case, the system can be represented as

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) + w, \tag{4.1}$$

where $\mathbf{x}_k$ represents the state of the system and $\mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k)$ describes the transition from one state to next. Transition of the state form $k-1$ to k can be evaluated by stepping/running the dynamic simulation forward by one time step.

For the convenience of expression, let us assume a vector $\mathbf{Z}_k$, which combines state and input vectors at time k. Let us assume another vector $\mathbf{Z}$ which contains the state and input of the whole trajectory until the prediction horizon N_p.

$$\mathbf{Z}_k = \begin{bmatrix} \mathbf{x}_k \\ \mathbf{u}_k \end{bmatrix}, \tag{4.2}$$

$$\mathbf{Z} = [\mathbf{Z}_0, ...\mathbf{Z}_{N_p}]. \tag{4.3}$$

Formation of the objective function or cost function is an essential part of the optimization process. After forming the cost function, the task of the optimizer is to maximize or minimize the cost function depending on the desired result from the system.

For the system represented by (4.1), a cost function can be formulated based on its states and corresponding inputs. The cost function of the system can be represented as

$$J(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{s}_k(\mathbf{x}_k) + \mathbf{i}_k(\mathbf{u}_k). \tag{4.4}$$

Here, $J(\mathbf{x}_k, \mathbf{u}_k)$ is the cost for the state at time k, $\mathbf{s}_k(\mathbf{x}_k)$ is the cost of the state and $\mathbf{i}_k(\mathbf{u}_k)$ is the cost of the input. However, cost can be interpreted as a probability using the following relationship [72]

$$r(\mathbf{x}_k, \mathbf{u}_k) \propto exp(-\mathbf{J}(\mathbf{x}_k, \mathbf{u}_k)), \tag{4.5}$$

where r is the probability of reaching the state to $\mathbf{x}_k$ for the input $\mathbf{u}_k$. Using the relationship in (4.5), the equivalent probability density function of the cost function in (4.4) can be formed. The equivalent probability density function for the trajectories until the prediction horizon is then

$$\mathbf{P}(\mathbf{Z}) = \frac{1}{M} \prod_k \Big[-\frac{1}{2} \Big(\mathbf{s}_k(\mathbf{x}_k) + \mathbf{i}_k(\mathbf{u}_k) \Big) \Big], \tag{4.6}$$

where M is a normalization factor. Moreover, the probability density function in (4.6) can be written in terms of potential functions [18] assuming the state and input potentials have maximum value at zero cost.

$$\mathbf{P}(\mathbf{Z}) = \frac{1}{M} \prod_k \alpha_k(\mathbf{x}_k)\beta_k(\mathbf{u}_k) \tag{4.7}$$

$$= \frac{1}{M} \prod_k \psi_k(\mathbf{Z}_k), \tag{4.8}$$

where α_k is the potential function of the state and β_k is the potential function of inputs and ψ_k is the potential function for the combined vector $\mathbf{Z}_k$. From (4.4) and (4.8), it is now evident that optimization problem can be solved by finding the maximum of the probability density function of the full trajectory $P(\mathbf{Z})$ instead of minimizing the cost function.

In (4.8), $\mathbf{P(Z)}$ shows the probability density of the full trajectory until the prediction horizon. On the other hand, in (4.8) all the $\mathbf{Z}_k$ are separate random variables at each node k. As we need valid and complete trajectories where each prediction step connects with the next prediction step, we need to add the probabilities of connecting two adjacent states in (4.8).

It is possible to form the transition potential from one state to another by finding probability density of states using (4.1). The probability density function of a vector-valued Gaussian random variable can be expressed as

$$\mathcal{N}(\mathbf{x}; \bar{\mathbf{x}}, \mathcal{P}) = |2\pi\mathcal{P}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\bar{\mathbf{x}})'\mathcal{P}^{-1}(\mathbf{x}-\bar{\mathbf{x}})} \tag{4.9}$$

where $\mathbf{x}$ is the random variable, $\bar{\mathbf{x}}$ is the mean and $\mathcal{P}$ is the variance. From (4.9) and (4.1) we can form the forward and backward transition potential as

$$\Psi_f(\mathbf{Z}_k, \mathbf{Z}_{k-1}) = \mathcal{N}\big(\mathbf{x}_k; f_k(\mathbf{x}_{k-1}, \mathbf{u}_k), \mathcal{P}\big), \tag{4.10}$$

$$\Psi_b(\mathbf{Z}_{k-1}, \mathbf{Z}_k) = \mathcal{N}\big(\mathbf{x}_{k-1}; f_k(\mathbf{x}_k, \mathbf{u}_k), \mathcal{P}\big). \tag{4.11}$$

where Ψ_f and Ψ_b are respectively forward and backward transition potential and $\mathcal{P}$ is the noise co-variance matrix ($w \sim \mathcal{N}(0, \mathcal{P})$). Using (4.10) and (4.11) it is possible to write

$$\Psi_f(\mathbf{Z}_{k-1}, \mathbf{Z}_k) = \Psi_b(\mathbf{Z}_k, \mathbf{Z}_{k-1}). \tag{4.12}$$

To ensure continuous trajectory (4.8) can be extended by using the forward and backward transition potential as

$$\mathbf{P}(\mathbf{Z}) = \frac{1}{M} \prod_k \psi_k(\mathbf{Z}_k) \prod_{k=1}^{N_p} \Psi_f(\mathbf{Z}_{k-1}, \mathbf{Z}_k) \prod_{k=0}^{N_p-1} \Psi_b(\mathbf{Z}_k, \mathbf{Z}_{k-1}), \tag{4.13}$$

where N_p is the prediction horizon. In Chapter 3, Markov random field was discussed, where (3.5) is a pairwise Markov random field. By comparing (4.13) and (3.5), it is evident that (4.13) represents a Markov random field, where any random variable $\mathbf{Z}_k$ is conditionally independent of other variables given the adjacent variables $\mathbf{Z}_{k-1}$ and $\mathbf{Z}_{k+1}$. The graphical representation for the Markov random field in (4.13) is shown in Figure 4.4.

The Markov random field constructed in the (4.13) can be written in more general form [6] as

$$P(\mathbf{Z}) = \frac{1}{M} \prod_s \psi_s(\mathbf{Z}_s) \prod_{(s,t)\in E} \Psi_{s,t}(\mathbf{Z}_s, \mathbf{Z}_t) \tag{4.14}$$

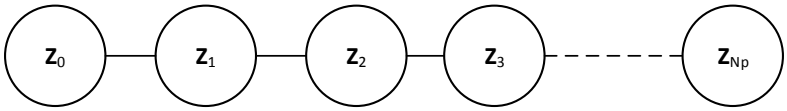


Figure 4.4: Markov random field.

where s is the current node, t is the target node and E is the set of the edges. In Chapter 3, it was discussed that particle belief propagation can be used for finding marginal probabilities of random variables using a graphical model. In this case, the graphical model expressed by (4.14) is used to find the marginal probability of producing a trajectory by the system, which will end up at certain position x_k for an input u_k.

For the graphical model shown in Figure 4.4, belief b_k at any node k can be expressed as [6]

$$b_k(\mathbf{Z}_k) = \psi_k(\mathbf{Z}_k) \prod_{s \in \Gamma_k} m_{s:k}(\mathbf{Z}_k), \tag{4.15}$$

$$m_{s:k}(\mathbf{Z}_k) = \sum_{\mathbf{Z}_s \in \mathcal{Z}_s} \Psi_{s,k}(\mathbf{Z}_s, \mathbf{Z}_k)\psi_s(\mathbf{Z}_s) \prod_{u \in \Gamma_s \setminus k} m_{u:s}(\mathbf{Z}_s), \tag{4.16}$$

where $\mathcal{Z}_s$ is the state space of $\mathbf{Z}_s$, Γ_s denotes the set of neighboring nodes of any node s and $m_{s:k}$ denotes message from s to k.

The operation of MPC can be described using two distinct time steps, control step and the prediction step. Control step n, refers to an optimized control action sent to control the real robot. For each control step, a number of trajectories are generated among which best trajectory is chosen and first control action is sent to the system. The prediction step k, is the simulation step for the prediction of the future states.

So far the graphical model only comprised of nodes that represent prediction steps. However, this graphical model can be extended by including both prediction time-step and control time-step. This will allow us to pass information between prediction and control time-step. The system equation can be extended as

$$\mathbf{x}_{k,n} = f(\mathbf{x}_{k-1,n}, \mathbf{u}_{k,n}) + w, \tag{4.17}$$

where k represents the prediction horizon time step and n represents the control step. The extended graphical model is shown in Figure 4.5. In this extension of the graphical model, the prediction time-step represents the best trajectory chosen for that particular control step.

In simulation, the initial state $\mathbf{x}_{0,n}$ for the trajectory prediction for a control step n can be found by using the first time step of the best trajectory of previous

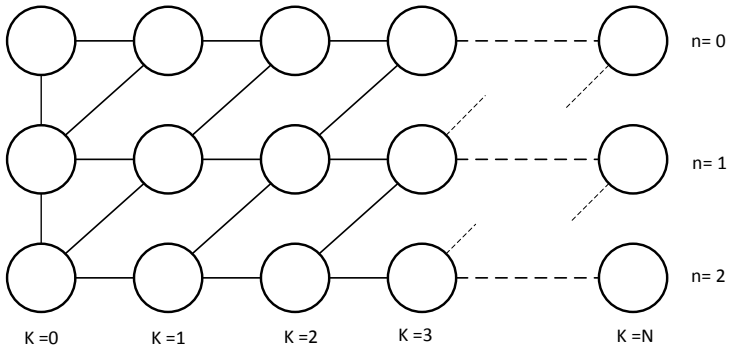


Figure 4.5: Extended graphical model [18].

best trajectory $n-1$.

$$\mathbf{x}_{0,n} = f(\mathbf{x}_{0,n-1}, \mathbf{u}_{1,n-1}) \tag{4.18}$$

So far, the algorithm is same as [18], where it was used for motion synthesis of 3D characters for animation. However, in this thesis, we are interested in implementing model predictive control for a real robotic system. The simulation process is fully deterministic as there is no uncertainty introduced due to the noise. On the other hand, due to the noise in the real system and the possible model mismatch, it is possible to have discrepancies between the simulation and the real system. To reduce the effect of the error between the simulation and the real world, a feedback is taken at each time step. This feedback is used as the initial position for predicting the future states at each control step.

Sample generation : To start the algorithm, a set of n number of samples are generated from an arbitrary proposal distribution $q_k(\mathbf{Z}_k)$, such that $\mathbf{Z}_k^i \sim q_k(\mathbf{Z}_k^i)$, where i is the sample index. This proposal distribution $q_k(\mathbf{Z}_k)$ can now be used to form an importance sampling weighted beliefs and messages. For particle belief propagation, the importance sampling corrected message and belief can be expressed as [6]

$$\hat{m}_{s:k}(\mathbf{Z}_k^i) = \frac{1}{N}\sum_{j=1}^{N} \Psi_{s,k}(\mathbf{Z}_s^j, \mathbf{Z}_k^i)\frac{\psi_s(\mathbf{Z}_s^j)}{q_s(\mathbf{Z}_s^j)} \prod_{u\in\Gamma_s/k} \hat{m}_{u:s}(\mathbf{Z}_s^j), \tag{4.19}$$

$$\hat{b}_k(\mathbf{Z}_k^i) = \frac{\psi_s(\mathbf{Z}_k^i)}{q_s(\mathbf{Z}_k^i)} \prod_{s\in\Gamma_k} \hat{m}_{s:k}(\mathbf{Z}_k^i). \tag{4.20}$$

If a graphical model is a simple chain or tree, the marginal probability distribution is proportional to the belief [6, 59]. In this case, the belief $b_k(\mathbf{Z}_k^i)$ is equivalent to the marginal probability of a generated trajectory by the system

to reach at x_k^i for an input u_k^i. This marginal probability of the samples are used to update the proposal density for the next step, resampling and also for smoothing operation for the trajectories.

As the messages are updated using importance sampling technique, it is necessary to form a proper proposal density because the accuracy of the estimation depends on how close the proposal distribution to the target distribution. The importance sampling technique is generally used when it is difficult to sample from the target density. To avoid this difficulty, samples are drawn from a proposal density from where it is much easier to generate samples. The purpose of importance sampling approach is to approximate target density using the samples generated from the proposal density. In this approach, each sample is given a weight based on the difference between the proposal and target density and the estimation is done based on the weights of the samples. However, in case of a high number of low weight samples, resampling can be performed. During resampling samples with low weights are discarded and more samples are generated from the high weight samples.

Let us generate a set of samples $\mathbf{Z}_k^i$ from the input potential $\beta(\mathbf{u}_k)^i$. The simulator can be used to find the state of the system for any given sample of the generates set. Using (4.1) we can write

$$\mathbf{x}_k^i = f(\mathbf{x}_{k-1}^i, \mathbf{u}_k^i) + w, \tag{4.21}$$

where k is the time step and i is the sample index. From (4.21), it is evident that it is possible to fully determine state $\mathbf{x}_k^i$ from the generated sample $\mathbf{u}_k^i$. As the vector $\mathbf{Z}_k^i$ consists of variable $\mathbf{x}_k^i$ and $\mathbf{u}_k^i$, where $\mathbf{u}_k^i$ fully determines $\mathbf{x}_k^i$, it can be said that the proposal density $q(\mathbf{Z}_k^i)$ depends only on $\mathbf{u}_k^i$ and can be expressed as

$$q(\mathbf{Z}_k^i) = \beta_k(\mathbf{u}_k^i). \tag{4.22}$$

As the samples are generated using the input potential $\beta_k(\mathbf{u}_k^i)$, it is important to form a proper input potential function by using the information of samples generated at the previous control step. Samples with high beliefs at the previous control step can be used to generate samples for the current control step. Referring to the extended graphical model in Figure 4.5, the samples for any control step n are generated by using samples from the previous control step $n-1$ as prior, where information passed from node $(k+1, n-1)$ to node (k, n). The control potential using the extended graphical model can be written as [18]

$$\begin{aligned}\beta_k(\mathbf{u}_{k,n}^i) = \mathcal{N}(\mathbf{u}_{k,n}^i; 0, \sigma_0^2\mathbf{C}_u)\ \ \mathcal{N}(\mathbf{u}_{k,n}^i; \mathbf{u}_{k-1,n}^i, \sigma_1^2\mathbf{C}_u) \\ \mathcal{N}(\mathbf{u}_{k,n}^i; 2\mathbf{u}_{k-1,n}^i - \mathbf{u}_{k-2,n}^i, \sigma_2^2\mathbf{C}_u) P(\mathbf{u}_{k,n}^i | \mathbf{Z}_{k-1,n-1}^i)\end{aligned} \tag{4.23}$$

where the first term of right-hand side of the equation is used for minimizing the control input and second and third term is for minimizing the first and second

derivative of the control input, where $\mathbf{C}_u$ is the diagonal covariance matrix. In this thesis, we use the angular velocity as u, to control the system. In this case, the angular velocity, acceleration and jerk can be minimized using σ_0, σ_1 and σ_2 respectively.

The information from previous control step is represented by the last term of the (4.23). Here, the control potential incorporates the information from the previous control step. Beliefs of the previously generates samples are used to generate new samples. In this case, a conditional Gaussian mixture model is generated to pass the information form the previous control step [18]. The Gaussian mixture model for passing information can be represented as

$$p(\mathbf{u}^i_{k,n}|\mathbf{Z}^i_{k-1,n-1}) \propto \sum_j^N w^j \mathcal{N}(\mathbf{u}^i_{k,n}; \mathbf{u}^j_{k+1,n-1}, \sigma_m^2 \mathbf{C}_u), \tag{4.24}$$

$$w^j = \hat{b}^j_{k+1,n-1}(\mathbf{Z}^j_{k+1,n-1}) \mathcal{N}(\mathbf{x}^i_{k-1,n}; \mathbf{x}^j_{k,n-1}, Q), \tag{4.25}$$

where w^i is the weight of the samples in the Gaussian mixture model and $b^j_{k+1,n-1}$ is the belief of the sample at the previous control step. The weights are calculated based on the belief of the samples in the previous control step.

Let us assume a N number of samples are generated from the proposal density $q(\mathbf{Z}^i_k)$. For these generated samples, it is possible to calculate the marginal probability by calculating belief using the graphical model. Using (4.19), (4.20) and (4.22), beliefs for the samples can be represented by (4.26)-(4.28) [18], where beliefs of the sample depends only on the state potential.

$$\hat{b}_k(\mathbf{Z}^i_k) = \alpha_k(x^i_k) \hat{m}_f(\mathbf{Z}^i_k) \hat{m}_b(\mathbf{Z}^i_k) \tag{4.26}$$

$$\hat{m}_f(\mathbf{Z}^i_k) = \frac{1}{N} \sum_{j=1}^N \Psi_f(\mathbf{Z}^j_{k-1}, \mathbf{Z}^i_k)\ \alpha_{k-1}(\mathbf{x}^j_{k-1})\ \hat{m}_f(\mathbf{Z}^j_{k-1}) \tag{4.27}$$

$$\hat{m}_b(\mathbf{Z}^i_k) = \frac{1}{N} \sum_{j=1}^N \Psi_b(\mathbf{Z}^j_{k+1}, \mathbf{Z}_k)\ \alpha_{k+1}(\mathbf{x}^j_{k+1}) \hat{m}_b(\mathbf{Z}^j_{k+1}) \tag{4.28}$$

where m_f is the forward message and m_b is the backward message. The graphical model is a simple tree, where any node can have maximum two neighboring nodes and at any instant, one node will receive messages from these two neighboring nodes. Using this convention the messages are named forward and backward messages.

Resampling: Resampling has an important effect on the accuracy of the estimation when the importance sampling technique is used. Without resampling, it is possible to have sample depletion, which means that after a few iterations, most of the samples may have very small or zero weight [73]. However, resampling also increases uncertainty for random sampling as it destroys information [73], which is why it is desirable to perform resampling only when it is needed for better estimation.

In this approach, resampling is done based on the forward belief. The forward belief for the system can be represented as

$$\hat{b}_f(\mathbf{Z}_k^i) = \alpha(\mathbf{x}_k^i)\hat{m}_f(\mathbf{Z}_k^i), \tag{4.29}$$

$$\hat{m}_f(\mathbf{Z}_k^i) = \frac{1}{N}\sum_{j=1}^{N} \Psi_f(\mathbf{Z}_{k-1}^j, \mathbf{Z}_k^i)\hat{b}_f(\mathbf{Z}_{k-1}^j). \tag{4.30}$$

As we have generated messages and beliefs based on importance sampling, forward belief in (4.29) essentially represents the weight of the sample. The number of effective samples (samples with good weight) can be calculated as

$$N_{th} = \frac{1}{\sum_i \left(\frac{\hat{b}_f(\mathbf{Z}_{k-1}^i)}{\sum_j \hat{b}_f(\mathbf{Z}_{k-1}^j)} \right)}. \tag{4.31}$$

Here N_{th} indicated the number of effective samples . This N_{th} can be used as a reference for performing resampling. Resampling can be performed when N_{th} is below a certain number, $N_{th} < TN$, where T can be used as a tuning parameter to set the threshold for resampling. As performing resampling in each iteration may result in loss of good samples, the proper choice of T is important for efficient estimation.

Smoothing: To ensure the best use of information produced during the prediction of states, a smoothing operation is done based on the beliefs of the samples generated during the forward pass. In general, smoothing refers to estimating the distribution of states at any time step, given that the information is available up to some later time steps. For example, to estimate the conditional distribution of $P(x_t|x_{1:l})$, for $l < t$, it is called the prediction of states. However, if $l > t$ the estimation is called smoothing. In this case, based on the belief on the forward pass, estimation at any time instant is done. For smoothing operation, more information is available than the prediction which generates smoother trajectory. Smoothing operation is carried out using backward passes on the graphical model. The starting point for the backward pass is estimated based on the state of the samples at the end of the prediction horizon as [18]

$$\hat{\mathbf{x}}_{N_p} = \frac{\sum_i \alpha(\mathbf{x}_{N_p}^i)\mathbf{x}_{N_p}}{\sum_i \alpha(\mathbf{x}_{N_p})}. \tag{4.32}$$

In this case, the starting point is the mean of the states at that time weighted by the state potential of the samples. Moreover, a Gaussian model is built based on the states $\mathbf{x}_{k+1}$, $\mathbf{x}_k$ and control vector $\mathbf{u}_k$. The Gaussian model is weighted using the forward belief for each sample which essentially maximizes the probability of the state to reach $\mathbf{x}_k$ for the control vector $\mathbf{u}_k$. The smoothed

state and control can be calculated as [18]

$$\hat{\mathbf{x}_k}, \hat{\mathbf{u}}_k = \mathbf{E}[\mathbf{x}_k, \mathbf{u}_k | \mathbf{x}_{k+1} = \hat{\mathbf{x}}_{k+1}], \tag{4.33}$$

$$\mathbf{E}[\mathbf{x}_k, \mathbf{u}_k | \mathbf{x}_{k+1} = \hat{\mathbf{x}}_{k+1}] = \mu_{\mathbf{x}_k, \mathbf{u}_k} + \mathcal{P}_{(\mathbf{x}_k, \mathbf{u}_k), \mathbf{x}_{k+1}} \big(\mathcal{P}_{\mathbf{x}_{k+1}, \mathbf{x}_{k+1}} + \lambda \mathbf{I}\big)^{-1} \big(\hat{\mathbf{x}}_{k+1} - \mu_{\mathbf{x}_{k+1}}\big). \tag{4.34}$$

where μ is the mean and $\mathcal{P}$ is the co-variance matrix. The information from smoothed trajectory obtained from this algorithm is also used to generate samples for the next time step.

Prediction and best control: The generated samples are used to simulate the system forward. As we have already formed the cost function in (4.4) it is now possible to optimize the control based on the cost function. For example, if N samples are generated at each step, we will have N trajectories until the prediction horizon and each trajectory has its own cost. These costs are then is transformed into a probability density function according to (4.8). Among all the generated trajectories, the best trajectory is

$$i_b = arg\ max_i(P(Z^i)). \tag{4.35}$$

where i_b is the sample index of the best trajectory. The first step of the best trajectory is then sent to the real robot. Information about the best trajectory is also used to modify the control potential for the next step. The first step of the best trajectory is also used by the simulator to move the system ahead for the next iteration. Moreover, the initial state for the next controls step is corrected based on the feedback from the real system. Algorithm 2 shows the complete algorithm for MPC combining simulation and optimization for a single control step.

Algorithm 2 MPC algorithm by combining optimization and simulation for a single control step n.

Require: Initial state $\mathbf{x}_{0,n}$(feedback from the real system), previous trajectories $\mathbf{u}^i_{1:N_p,n-1}$, previous best trajectory $\mathbf{u}^b_{1:N_p,n-1}$, smoothed trajectory $\hat{\mathbf{u}}_{1:N_p,n-1}$, prediction horizon N_p, number of samples N, resampling threshold T, cost function J_k.

Ensure: Optimized control sequence, sampled trajectories.

Correct the initial state in the simulator using feedback from real system.
Save the current ODE state. (Master state)
for k = 1....N_p **do**
  Compute N_{th} of of the previous step. (4.31)
  if $N_{th} < TN$ **then**
    Resample
  end if
  for i= 1...N **do**
    if i==1 **then**
      $\mathbf{u}^i_{k,n} = \mathbf{u}^b_{k+1,n-1}$
    else if i ==2 **then**
      $\mathbf{u}^i_{k,n} = \hat{\mathbf{u}}_{k+1,n-1}$
    else
      Generate sample $\mathbf{u}^i_k$. (4.22), (4.23)
    end if
    Simulate system forward using $\mathbf{u}^i_k$.
  end for
  Compute forward beliefs. (4.29),(4.30)
end for
Smoothing operation. (4.32)-(4.34)
Find optimal control. (4.35)
Restore the Master State.
Deploy optimal control in simulator. (4.18)
Deploy optimal control to the robot.
return: best trajectory $\mathbf{u}^b_{1:N_p,n}$, generated trajectory $\mathbf{u}^i_{1:N_p,n}$ and smoothed trajectory $\hat{\mathbf{u}}_{1:N_p,n}$.

Chapter 5

Robot Tray Balancing

In the previous chapter, the general algorithm for a MPC combining simulation and optimization was described. In this chapter, details of the specific implementation of the proposed algorithm on a robotic system are discussed. A specific robotic system, which is essentially nonlinear in nature was built using a robotic arm to implement the proposed algorithm and to test its functionality.

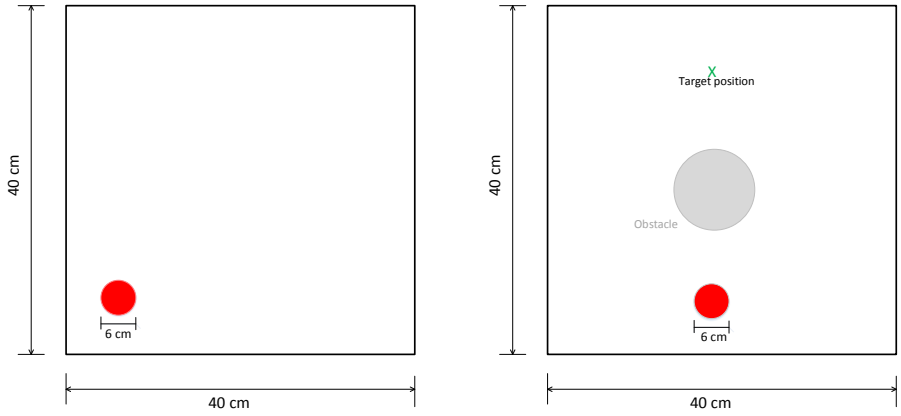
5.1 Task description

The robotic system under consideration consists of a freely moving object and a flat platform attached to a robotic arm. The trajectory of the moving object can be controlled by controlling the movement of the platform using the robotic arm. Figure 5.1 shows the design of the platform which is attached to a robotic arm and the object, a ball, is placed on the platform.

The task of the controller is to generate a trajectory, such that the object is moved from any starting position to a user defined final position and balanced on the target position. The movement of the ball is controlled by manipulating the platform using the robotic arm. Two joints of the robotic arm are used to rotate the platform about two different axes using angular velocity as the control parameter.

To test the functionality of the designed controller, three different tasks are considered. The first task is to control the ball in only one axis, whereas the second task is to control the position of the ball in two axes using two joints of the robotic arm. The final task is to control the robot with a presence of constraint. Figure 5.1(a) shows the setup for the first two tasks. On the other

hand, Figure 5.1(b) shows the setup for the final task, where an obstacle is placed between the starting and the target point.



(a) (b)

Figure 5.1: Design of the platform (a) for the first two tasks, (b) for the final task.

For this robotic system under consideration, the state vector consists of two variables, x and y-axis position of the ball on the platform. The state vector can be represented as

$$\mathbf{x}_k = [x_k \; y_k]^T, \tag{5.1}$$

where x_k is the ball position in the x-axis and y_k is the ball position in the y-axis. Similarly, the control vector also consists of control angular velocities for two joints. The control vector can be written as

$$\mathbf{u}_k = [u1_k \; u2_k]^T, \tag{5.2}$$

where $u1_k$ is the control angular velocity of first joint and $u2_k$ is the control angular velocity of the second joint. However, for the first task only one of the state variable and the corresponding input variable is considered.

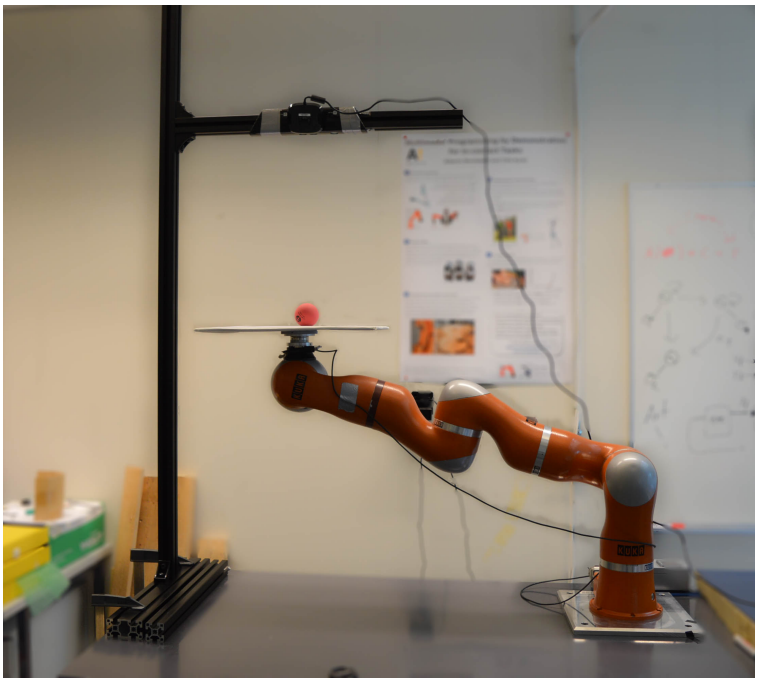
The cost function of the current system is formulated based on the equation 4.4. In this case, cost function can be represented as

$$J(\mathbf{x}_k) = (x_t - x_k)^2 + (y_t - y_k)^2, \tag{5.3}$$

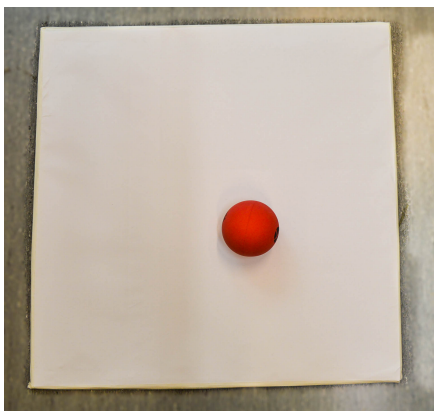
where x_t is the target position in the x-axis and y_t is the target position in the y-axis. Each generated trajectory is associated with a cost based on (5.3). In (5.3), only state variables are used to form the cost, but it is possible to include other parameters in the cost function, such as control parameters (angular velocity of joints) and linear velocity of the ball. Both approaches of forming the cost function are used to test the effect of including different parameters in the cost function. However, for the first task only one of the state variable and the corresponding input variable is considered.

5.2 Hardware

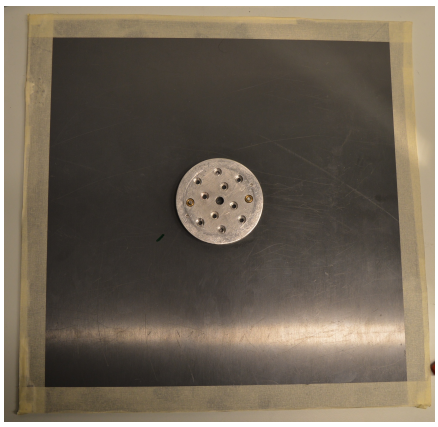
Hardware of the implemented system can be divided into five distinct parts: 1. The KUKA lightweight robotic arm, 2. Platform, as the end-effector, 3. The ball, which is the to be controlled, 4. Camera, to provide feedback 5. External computer, to run the simulations and optimization process. Figure 5.2 shows the hardware overview of the system (without the external computer).



(a)



(b)



(c)

Figure 5.2: (a) Hardware of the system except the external computer. (b) Platform and the ball (front view). (c) Platform (bottom view).

The KUKA LWR 4+ was used to manipulate the end-effector. It has seven

revolute joints controlled by harmonic drives and compact brushless motors, which allows the joints to rotate in seven different axes. For the current tasks, the rotation of the platform around two axes from a fixed point of origin needs to be controlled. Figure 5.3 shows the coordinate system used for this system and Figure 5.4 shows the joints used to control the orientation of the platform. The joint *a5* is responsible for rotating of the platform about x-axis and joint *a6* rotates the platform about y-axis.

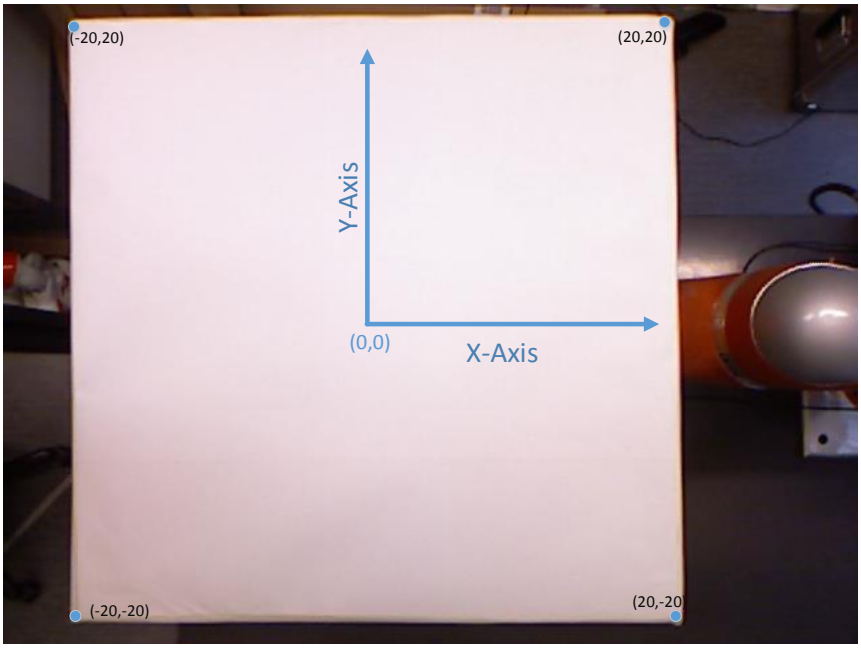


Figure 5.3: Coordinate system of the system.

The rotation of the joints was controlled by using joint angular velocity. In addition, using KUKA LWR 4+ it is possible to change the stiffness of the arm. However, to make the simulation model simpler, a high stiffness value was used which remained constant throughout the operation. The desired velocity of the joints was applied using joint specific position control of the KUKA, using the following equation:

$$\theta = \dot{\theta} \times t, \tag{5.4}$$

where θ is rotation in radian, $\dot{\theta}$ is angular velocity (rad/s) and t is the time step.

The hardware configuration of KUKA LWR can be divided into two parts, the robot arm and KRC. KRC is an industrial computer used to control the arm. It is possible to communicate between the KRC and an external computer via Ethernet connection using a User Datagram Protocol (UDP) [74]. For this work, simulation and optimization were carried out on an external computer and optimized controls were used to control the motion of the robotic arm from the external computer.

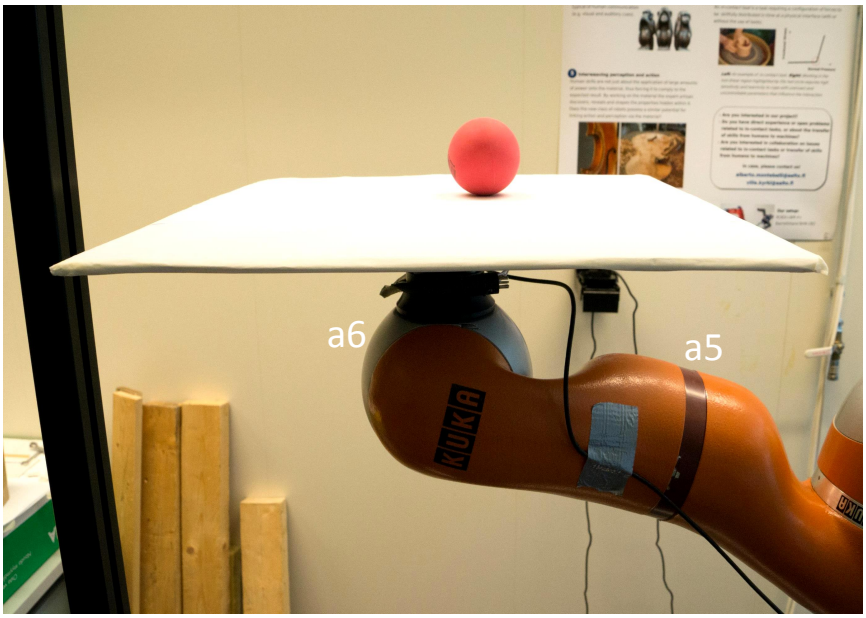


Figure 5.4: Joints used to control the orientation of the platform.

The platform attached to the robotic arm is a rectangular plate, made of plastic and the moving object is a red rubber ball. The parameters of the platform and ball are shown in Table 5.1.

	Platform	**Ball**
Dimension	$40cm \times 40cm$	-
Radius	-	$3cm$
Weight	$0.98kg$	$0.04kg$

Table 5.1: Parameters of the platform and the ball.

To provide feedback about the position of the ball on the platform, a camera based feedback system was built. A Microsoft KINECT depth camera was used to get the position of the ball in the fixed coordinate system. The camera was fixed on top of the platform and calibrated to give the position of the ball on the platform.

An external computer was used to control the whole operation of the system. Evaluation of the processing time is important for the system, as the ability of real-time controller depends on the processing power of the computer. The external computer used has 4 physical processors with an operating frequency of 3.3 GHz.

5.3 Simulation model

As discussed in the earlier chapters, a realistic and accurate simulation model of the system is important for designing a model predictive controller as it replaces the mathematical model of the system. Based on the task of the system, a simplified simulation model of the KUKA arm was built. Figures 5.5 and 5.6 show the model that was used for simulation. In this model, two joints a5 and a6 are simulated using hinge type joints. A hinge joint allows rotation only about one axis between the connected bodies.

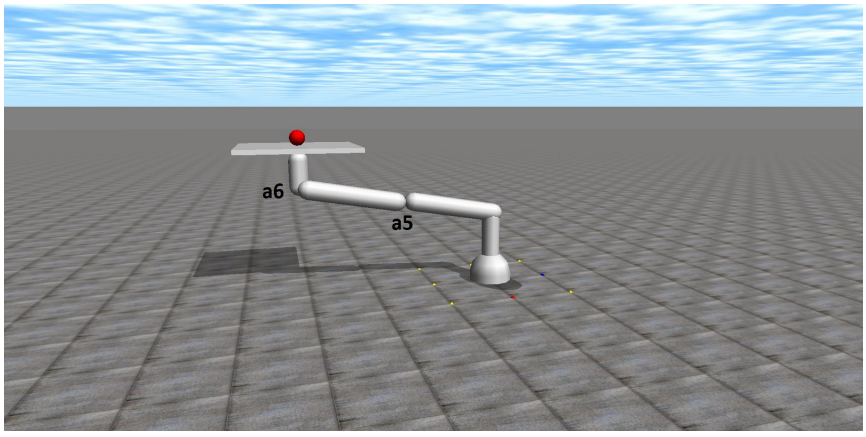


Figure 5.5: Simulation model (side view).

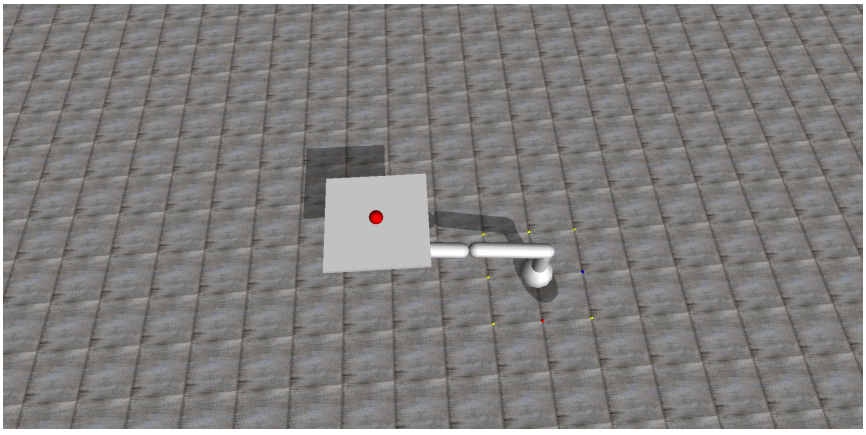


Figure 5.6: Simulation model (top view).

An OpenGL library was used to visualize the simulation. However, visualization of the simulation is computationally expensive and not possible to implement in real-time with the optimization technique involved. Nonetheless, visualization of the simulation is not important for this work and was only used for documentation purpose.

5.4 Software architecture

To communicate with the robot arm and control its motion from an external computer, a software package called Fast Research Interface(FRI) is available [74]. FRI is the software add-on that enables the communication between and external computer and the KRC. Using FRI, it is possible to read robot system data and execute custom programs to control the robot from the external computer. Figure 5.7 shows the control architecture for the FRI.

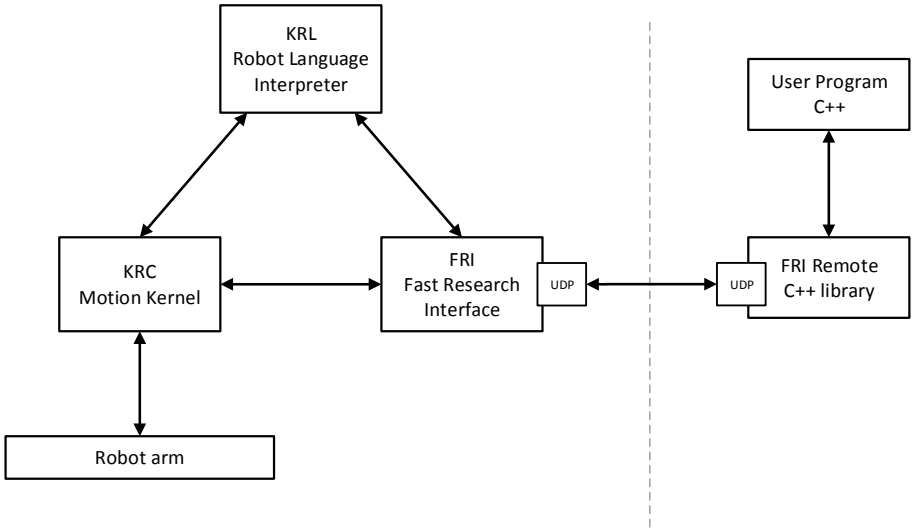


Figure 5.7: Control architecture for the FRI.

A basic framework for controlling the robot using FRI was already implemented in the KUKA used [75]. OROCOS and ROS were used as middleware for the communication using FRI. OROCOS is a free software framework developed for multipurpose control of robotic systems [76]. Though OROCOS is flexible and capable of running on any platform, the most important properties of OROCOS is its real-time capability. The operation in OROCOS is based on components, which can be used for real-time applications. Every component can have data flow ports, by which it is possible to exchange data between the components. Figure 5.8 shows typical data flow connections between OROCOS components.

On the other hand, ROS is an open-source programming framework designed for a range of robotic applications [77]. ROS has become very popular within a very short period of time since its unveiling in 2007. It can be labeled a meta-operating system for robotic systems. It provides standard operating system functionality, such as hardware abstraction, low-level device control, message-passing between processes, and package management [77]. Moreover, it also provides some high-level functionality, such as asynchronous and synchronous

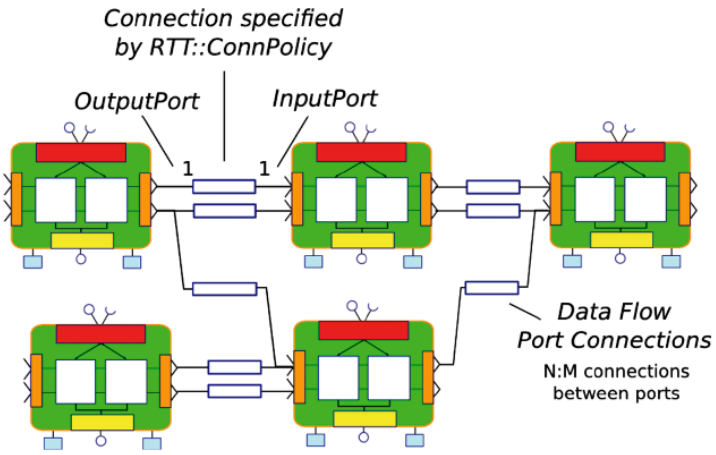


Figure 5.8: Typical data flow connections between OROCOS components [76].

calls, centralized database and a robot configuration system. In general, a system in ROS can be represented by a number of independent nodes. These nodes communicate with each other by passing messages with a certain topic name.

Due to the integration of ROS and OROCOS it is possible to share messages between them. Data published from a ROS node via a topic can be read from an OROCOS port and vice versa.

Figure 5.9 shows the basic framework provided for communication between KRC and an external computer using FRI. `FRIServerRT` is an OROCOS component, which is the gateway between the KRC and the external computer. The robot data from the KRC is collected by this component, which is then written to its output ports. These data can be used by other components after connecting the input port of the second component to the output port of the `FRIServerRT`. Moreover, it can also be used to send data to KRC to control the robot motion and stiffness. In this case, the output port of the second component needs to be connected to the input ports of the `FRIServerRT`.

In this thesis work, a new OROCOS component `Commander` was created, which is responsible for collecting only the relevant data from `FRIServerRT` and sending the control data to `FRIServerRT`. Moreover, this component communicates with a ROS node to collect the final control command, that needs to be sent to the robot.

The system has three ROS nodes responsible for different operations: 1. A camera node that runs the camera module and provides the position of the ball 2. A simulation node that runs the simulation and optimization process 3. A control node that collects the data from the camera, robot and simulation

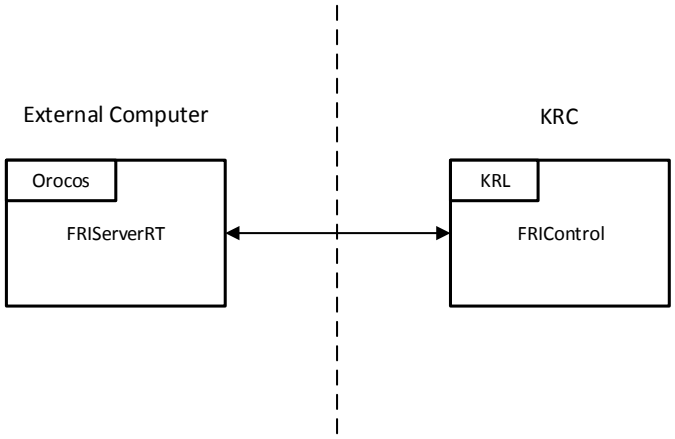


Figure 5.9: Provided framework for robot control.

node. The control node handles the information from all the nodes and sends the processed information to the relevant nodes and OROCOS component. A complete overview of the software architecture is shown in Figure 5.10.

Camera node: The camera node provides the feedback about the ball position on the platform. It uses OPENNI with OPENCV to collect and process images. OPENNI is an open source SDK, which provides the framework for initializing and collect data from depth sensors [78]. On the other hand, OPENCV is an open source library for image processing and computer vision [79]. A ROS package *cv_bridge* was used for the OPENCV library, which provides the interface between ROS and OPENCV.

Camera position is fixed and calibrated to provide the position of the ball in two dimensions with respect to the fixed coordinate system. For detecting the ball, a color-based method was used. To have a significant difference in color between the object and the background, a red ball was used, while the platform was white. After processing the image of the ball on the platform, the calculated position of the object was published via a ROS topic.

Simulation node: This node is responsible for running the simulation and carries out the optimization based on the simulation. This node is part of a request/response ROS-service, where it acts as a server. It receives the request from the control node along with the current ball position. In turn, it responds with the optimized control signal.

Control node: This node synchronizes the operation of the whole system. It is connected to all other ROS nodes and the OROCOS component. It collects information from the camera node as well as the joint position of the KUKA arm from the OROCOS component `Commander`. It is a part of the same request/response service used by the simulation node, where it acts as the client.

It sends the request with the current position of the ball in the real system and collects the response from the simulation node in the form of the final control signal. Collected final control signal is then sent to the `Commander` using a ROS topic.

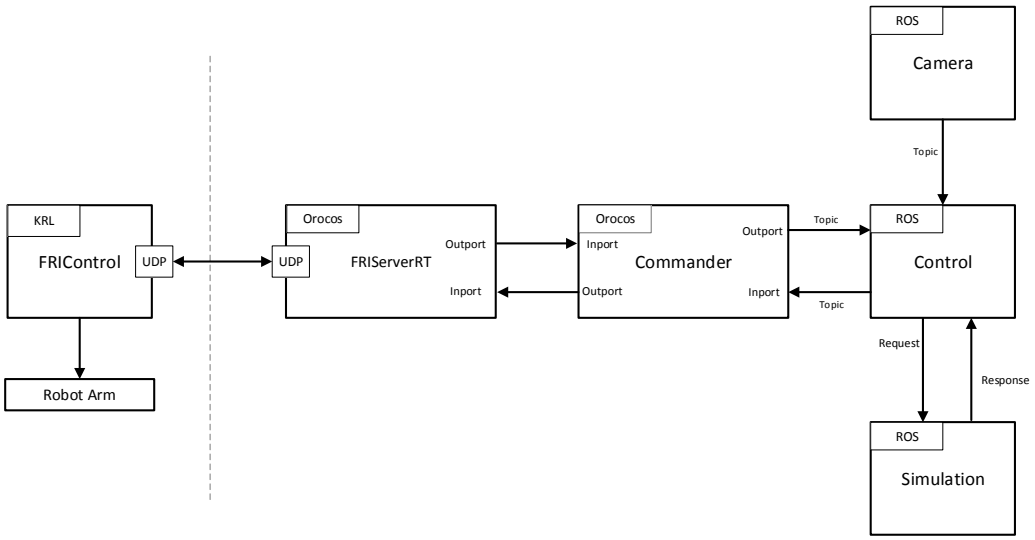


Figure 5.10: Software architecture of the system.

5.5 Operation

The data transfer between the KRC and external computer using FRI can be done in two distinct modes, monitor mode and command mode. In monitor mode, only the robot data can be read by the external computer. However, in command mode in addition to reading the robot data, robot motion and position can also be controlled externally. Moreover, to control the position, motion and stiffness of the arm, KUKA has three different controllers, position controller, Cartesian stiffness controller and axis-specific stiffness controller. For the operation of the tasks, command mode and the joint position controller was used throughout the operation.

After setting up KUKA in command mode and joint position controller mode, all the OROCOS components and ROS nodes are started. The camera node calculates the position of the ball and sends it to the control node. The control node receives the position data and sends it to the simulation node as a request and wait for the response from the simulation node. On the other hand, simulation node receives the position data and set the ball position as the starting point in the simulation model. This is done in every iteration, which essentially

corrects position in the simulator in every step based on the feedback from the real system. After that simulation node carries out the simulation and optimization using the method discussed in algorithm 2 in chapter 4. In Figures 5.11 and

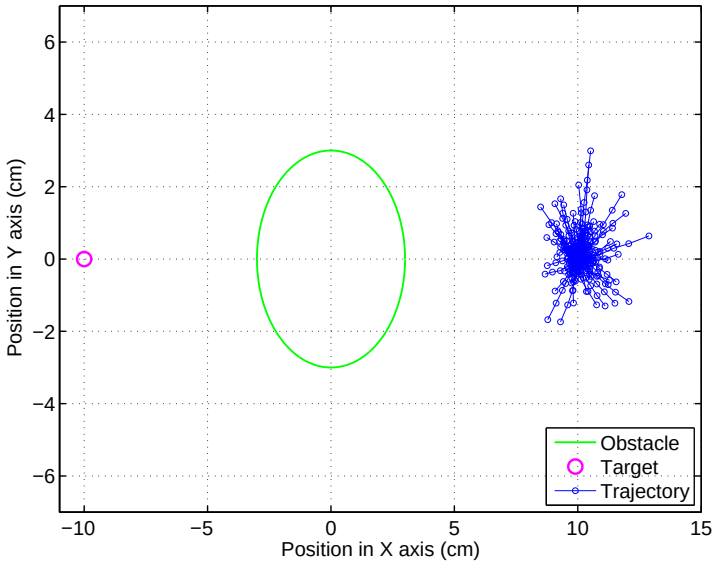


Figure 5.11: Predicted trajectories at the beginning of the operation.

5.12, examples of generated trajectories are shown, where 50 samples were generated at each time step and prediction horizon was 10 time steps. Figure 5.11 represents the generated trajectories for the samples generated at the beginning of the optimization process, where no previous information was available. On the other hand, Figure 5.12 shows the generated trajectories after a few control time steps. For the trajectories shown in Figure 5.12, the best trajectory was chosen based on the cost function, which is marked in red.

When the simulation node finishes the calculation and finds the best controls for the system, it responds to the request of the controller node and sends the control information to the control node. Control node then passes it to `Commander`, which is then sent to KUKA via `FRIServerRT`. To start the next iteration, the simulation node again receives the ball position and set this value as the starting position for the next control step.

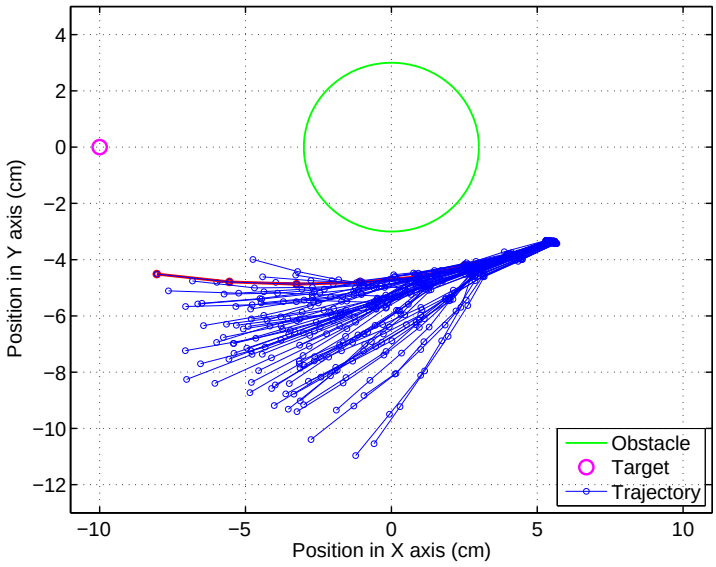


Figure 5.12: Predicted trajectories after few control step. Trajectory shown in red represents the chosen best trajectory.

Chapter 6

Experiments and Results

This chapter presents experiments and their corresponding results that were performed to test the functionality of the designed model predictive controller. The quality of the simulation was evaluated and based on the experiment proper simulation parameters were chosen. The capability of the designed controller to handle a nonlinear system with multiple variables and constraints were evaluated.

6.1 Quality of simulation

The objective of this experiment was to analyze the quality of the simulation with respect to the real system. The accuracy of the physics engine was evaluated in this test and appropriate simulation parameter values were chosen.

For this experiment, the robotic system discussed in Chapter 5 was used. Joint $a5$ was used to rotate the platform, to move the ball along the x-axis. A constant angular velocity of 0.1 rad/s was used to rotate the platform. To compare the similarity between real world and the simulation, the same operation was done in the simulator, where the platform was rotated using the same constant joint angular velocity of the corresponding joint and the position of the ball was recorded at each time step.

The accuracy of the simulation depends on certain simulation parameters. The parameters that affect the simulation of any physical system can be divided into two categories: global simulation parameters of the simulator and parameters attributed to physical properties of the system. Global simulation parameters are the most important and easily tunable. In general, simulation in ODE focuses more on high performance than accuracy [71]. However, the accuracy

of the simulation can be increased by tuning these parameters. Among the global simulation parameters, most relevant to this work are, simulation step size, Error reduction parameter (ERP), Constraint force mixing (CFM). On the other hand, The second category includes a number of parameters, such as the shape, mass, bounce and friction between the objects.

Friction coefficient: The shape and mass of the objects are fixed and bounce has no significant effect on the operation of the system, which makes the friction between the objects an important parameter. To approximate friction between contact points, ODE uses Coulomb friction model [71]. Using this model, the simulator calculates the force magnitude and direction at each contact point. ODE calculates the contact force as

$$\mathbf{F}_m = \mu \mathbf{F}_N, \tag{6.1}$$

where F_N is the normal component of the force between the bodies and F_m is the maximum limit of the frictional force and μ is the friction coefficient. As we are simulating a ball, rotating on a flat platform, the force-induced torque for the rotating ball is calculated in ODE as

$$\tau = \mu \mathbf{F}_N \sum \mathbf{r}_i, \tag{6.2}$$

where $\mathbf{r}_i$ is the distance of contact point from the rotating center.

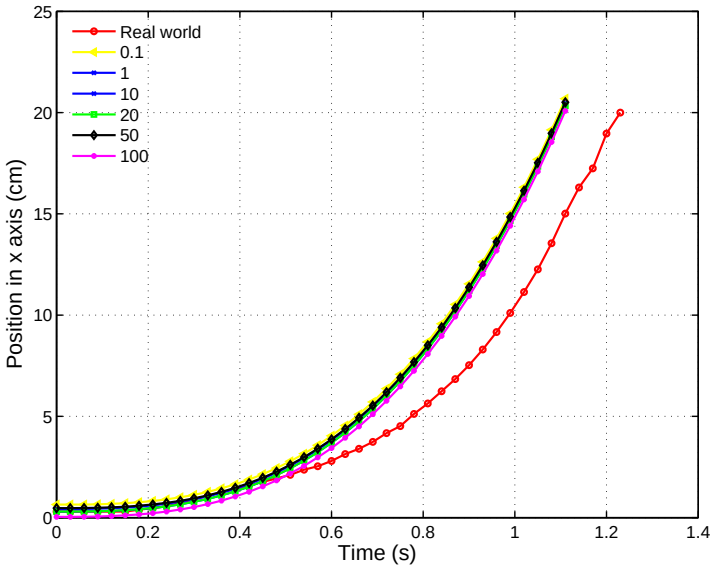


Figure 6.1: Comparison of ball positions for different friction coefficient.

To find out the effect of friction coefficient on the simulation, tests were carried out for different friction coefficient values. Figure 6.1 shows a comparison between the real system and the simulated system for different friction coefficient values. From the Figure 6.1, we can conclude that for the current system the

friction coefficient does not have a significant effect on the accuracy of the simulation. In this experiment, a range of values from 0.1 to 100 were chosen and the results suggest that for all the values, the simulation provided the similar results.

Simulation step size: Another important parameter for accurate simulation is the simulation step size. Simulation step size defines how much the system will move forward by the simulator. ODE uses a semi-explicit Euler integrator for numeric integration [71]. Due to the use of Euler integration, the time step size of the simulation is an important factor as the accuracy of the simulator decreases with the increase in time step size.

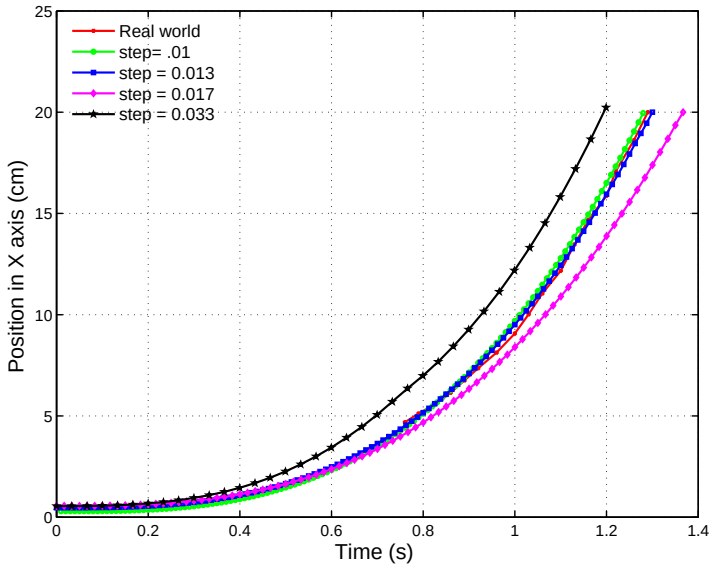


Figure 6.2: Comparison of ball positions for different time step size.

Figure 6.2 shows that accuracy of the simulation depends on step size. However, even though step size of 0.033 s did not provide the best result, it was chosen because of the limitation of the camera feedback system used. The KINECT depth camera used for the operation can operate at maximum 30 fps [80]. The minimum time between two measurements of the depth sensor is 0.033 s. To avoid synchronization error between the real system and the simulation, step size of 0.033 s was chosen. Even though the bigger step size provides relatively higher error, the accuracy of the simulation can be improved by tuning other simulation parameters, such as error reduction parameter (ERP) and constraint force mixing parameter (CFM).

Error reduction parameter (ERP): In ODE, to connect two or more rigid bodies together, different joints can be used. Joints in ODE correspond to constraints on the position and orientation between the connected bodies [71]. These constraints specify relative position and orientation between the bodies.

However, these constraints may get violated due to numerical errors during the integration phase of the simulation. One of the easiest way to reduce the constraint violation error is to use smaller step size for simulation. Another way of reducing constraint violation is using error reduction parameter (ERP). In ODE, it is possible to correct constraint errors to some extent by using the error reduction parameter. The error correction procedure in ODE involves applying a joint specific force to restore the proper alignment of the bodies. This force is controlled by the error reduction parameter. The ERP is generally a value between 0 to 1, specifies the proportion of the joint error to be fixed during the next simulation step. However, a higher value of ERP is not desirable as it introduces damping in the system [81]. The system becomes fully damped if ERP =1 is used. Figure 6.3 shows the comparison between the trajectories for different ERP values for simulation step size of 0.033 s.

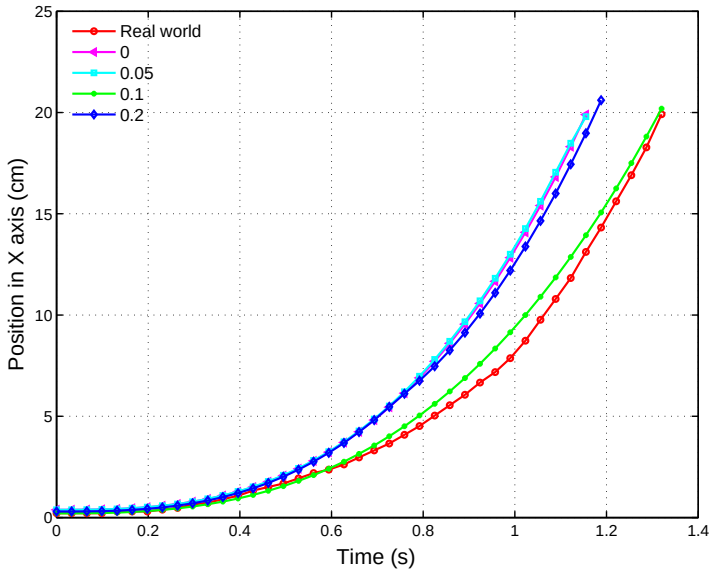


Figure 6.3: Comparison of trajectories for different ERP values. (Step size = 0.033 s).

Figure 6.3 suggests that ERP has a significant effect on the accuracy of the simulation. In this case, the tests were carried out with the time step of 0.033 s, which provided higher error without using ERP, shown in Figure 6.2. However, with ERP = 0.1, it provides a significant improvement in the accuracy of the simulation.

Constraint force mixing (CFM) : In general, ODE uses hard constraints for its operation, which means that the constraints are not allowed to be violated. The use of hard constraints essentially suggests that no interpenetration between the objects are allowed. However, we have used a rubber ball which is soft and allows some natural penetration. Fortunately, ODE provides provisions for violating the hard constraints using the parameter CFM.

The CFM can be used to simulate contacts between softer objects where there is some natural penetration between objects, when they are forced together. In general, constraints for a joint can be expressed as

$$Jv = c \tag{6.3}$$

where J is the Jacobian matrix and v is the velocity. However, in ODE the constraint equation has the following form

$$Jv = c + k_{CFM}\lambda, \tag{6.4}$$

$$F = J^T\lambda \tag{6.5}$$

where F is the force which is applied to enforce the constraints. The vector λ is calculated using the constraint force F. In this case, nonzero value of k_{CFM} allows the violation of constraint proportional to k_{CFM}.

To evaluate the effect of CFM on simulation, test were carried out with different CFM values. Figure 6.4 shows the comparison of trajectories for different CFM values for step size 0.033 s and ERP=0.1.

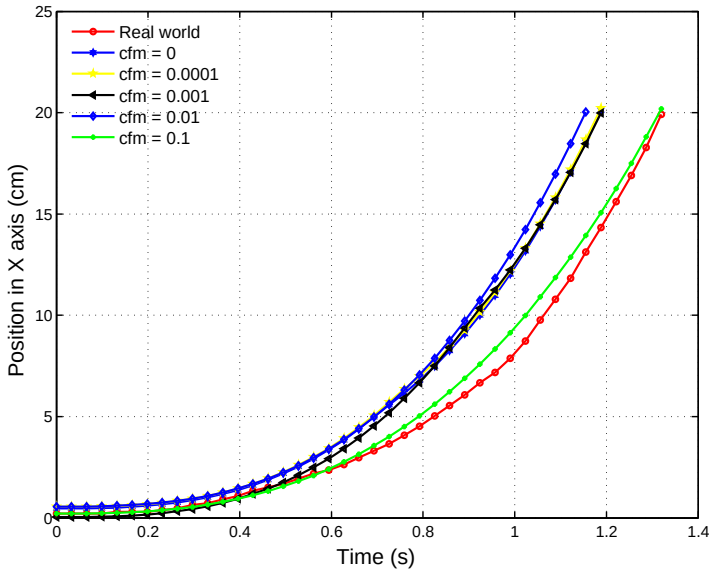
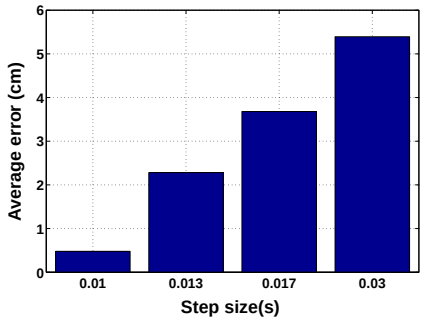


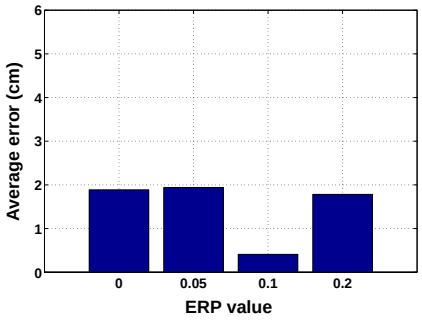
Figure 6.4: Comparison of trajectories for different CFM values (step size= 0.033 s. ERP = 0.1).

Figure 6.4 shows that the proper choice of CFM value is important as it has a significant effect on the accuracy of the simulation. From this experiment, the CFM value which provided the highest accuracy was 0.1.

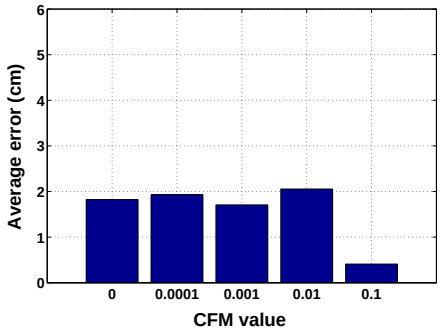
Based on the above tests, it is evident that the choice of simulation parameter values is important for an accurate simulation of the system. Figure 6.5 shows the average errors for all the experiments. From the Figure 6.5(a) we can



(a) For ERP = 0, CFM = 0



(b) For step size = 0.033 s, CFM = 0.1



(c) For step size= 0.033 s, ERP = 0.1

Figure 6.5: Average error in simulation (a) for different step size, (b) for different ERP values, (c) for different CFM values.

see that average error for time step size 0.033 s is relatively high, but from Figures 6.5(b) and 6.5(c), it is apparent that accuracy for simulation even for higher simulation step size can be increased by using proper ERP and CFM values. For the current system, best parameter values chosen for simulation is shown in Table 6.1, for which the average error is 0.4 cm.

To test the consistency of the results, total 20 trials of the same operation were carried out using the same simulation parameters in Table 6.1. From the results of trials, it was found that the average error for each trial was between 0.4 to 0.5 cm.

Moreover, to check the reliability of the chosen simulation parameters same operation was carried out in y-axis, which showed the similar results. Among 20 trials in y-axis, the average error was found between 0.38 to 0.45. However, the reliability of the chosen simulation parameter values were not tested with different angular velocities.

Parameter	Value
Time step size	0.033 s
Friction coefficient	1
Error reduction parameter (ERP)	0.1
Constraint force mixing (CFM)	0.1

Table 6.1: Chosen parameter values from the experiment.

6.2 Robot tray balancing: Single variable control

The objective of this experiment was to evaluate the capability of the controller for a nonlinear system with a single variable. In this experiment, the system described in Chapter 5 was used, where the motion of the ball was controlled in only one axis. The task for this test was to move the ball from any starting point to a user defined target point and balance the ball at the target point. The optimization parameters used for this experiment are shown in the Table 6.2 and the simulation parameters used are shown in Table 6.1.

Parameter	Value
Number of trajectory per time step, N	50
Prediction Horizon, N_p	15 time step (0.45 second)
Resampling Threshold, T_{th}	0.5

Table 6.2: Parameters used for optimization.

Figure 6.6 shows the position of the ball in x-axis with respect to time step. In this case, the user defined target position was x = 0. From the figure, it is apparent that the controller was capable of balancing the ball close to the target. However, there was some offset between the desired target point and the achieved final point. The probable reason behind the offset is the lack of accurate measurements from the feedback system, which is mainly due to the uncertainty in calculating the center of the ball.

To test the capability of the controller, further tests were made for different target positions. Figure 6.7 shows the result of another test, where the target position was x= 3. Moreover, to test the consistency of the controller, test were carried out also for the y-axis. Figure 6.8 shows the trajectory of the ball in the y-axis, where the target was y = 0.

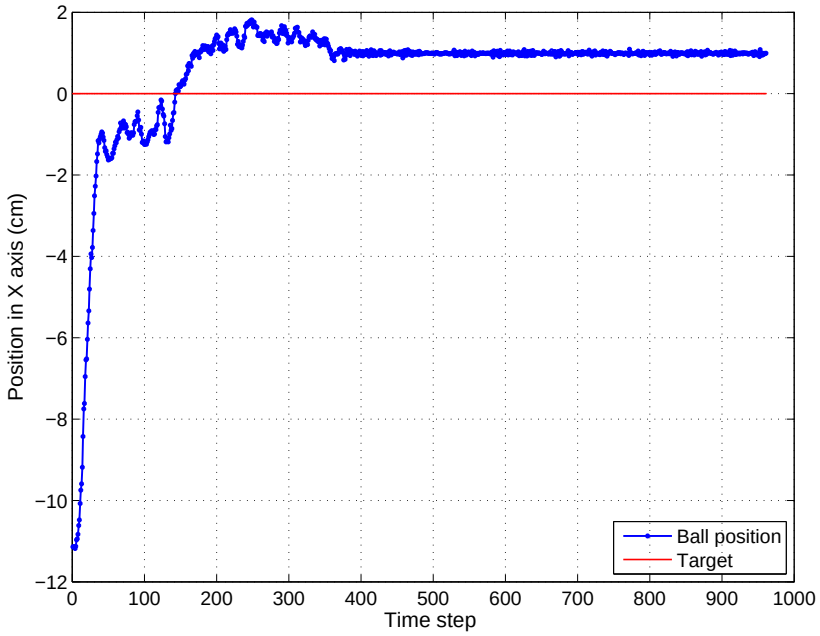


Figure 6.6: Single axis control (target x=0).

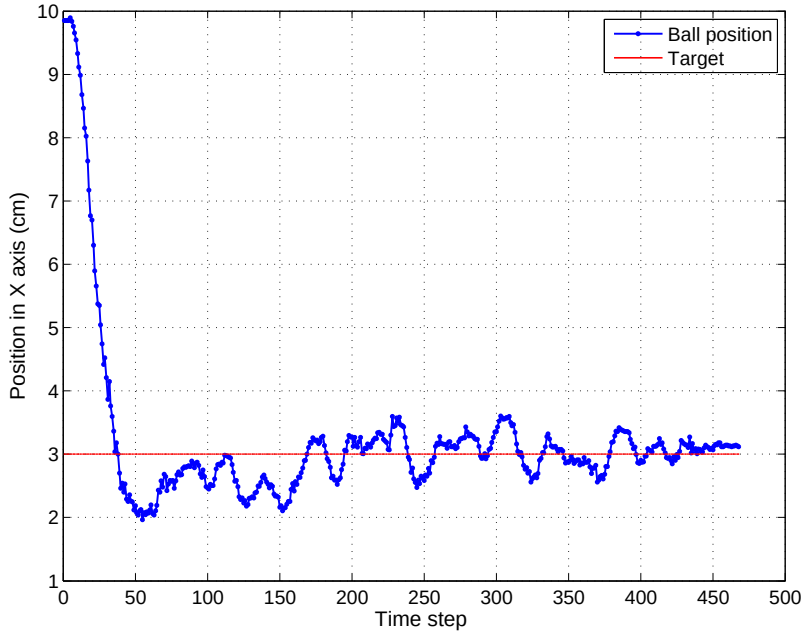


Figure 6.7: Single axis control (target x=3).

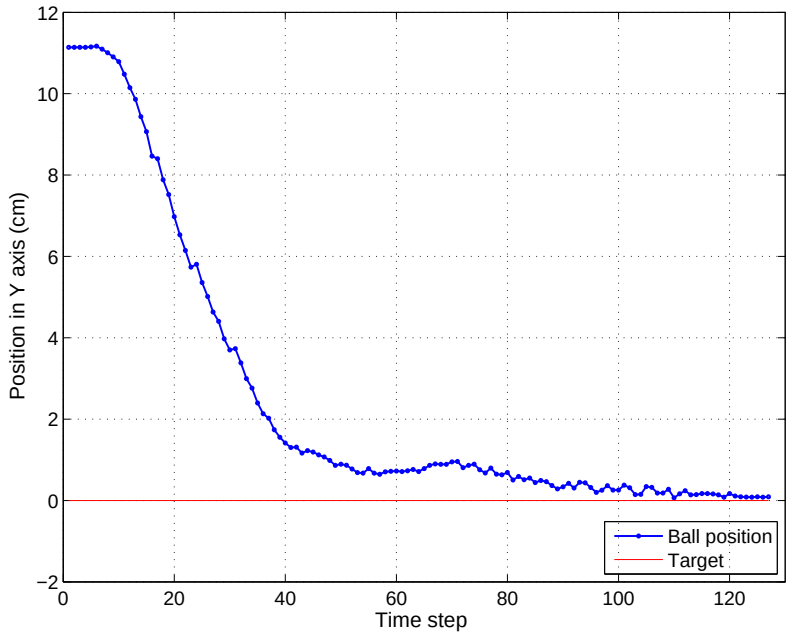


Figure 6.8: Single axis control (target y=0).

6.3 Robot tray balancing: Multivariable control

One of the important advantages of MPC over other conventional technique is its ability to handle multiple variables. The purpose of this test was to evaluate the capability of the proposed controller to work with multiple variables in real time. Another important aspect of MPC is that it provides the opportunity to form and modify the cost function intuitively. The effect of the modification of the cost function was also evaluated in this experiment.

In this test, the aim was to move the ball from a starting position (x, y) to the a user defined target position(x_t, y_t) using both the joints $a5$ and $a6$. For this experiment the same optimization parameters in Table 6.2 and simulation parameters of Table 6.1 were used. Figure 6.9 shows the trajectory of the ball for the current task. In this case, the target coordinates were (0,0). To test the reliability of the controller, the test was carried out for different starting position. Figure 6.10 shows the trajectory, where the task was carried out with a different starting position.

However, for the experiment results shown in Figures 6.9 and 6.10, cost function included only state vectors, the position of the ball in x and y-axis. Other state and control parameters can also be included in the cost function. For example, the control parameters, angular velocities for the joints can also be included in

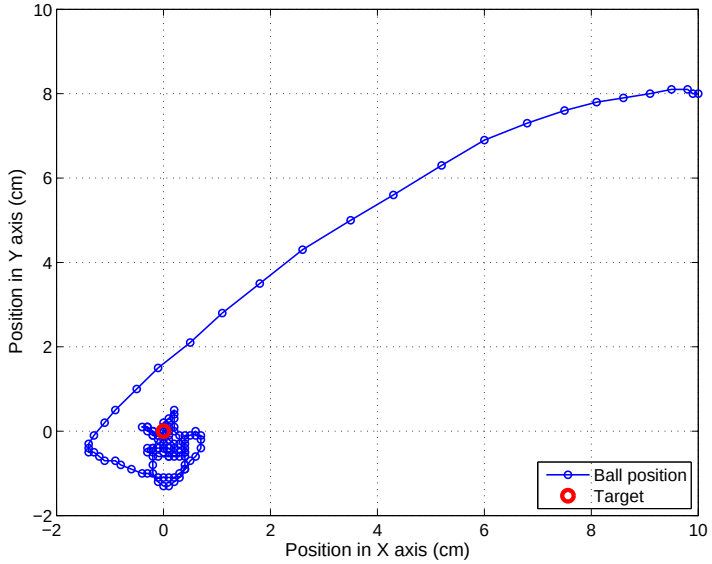


Figure 6.9: Double axis control.

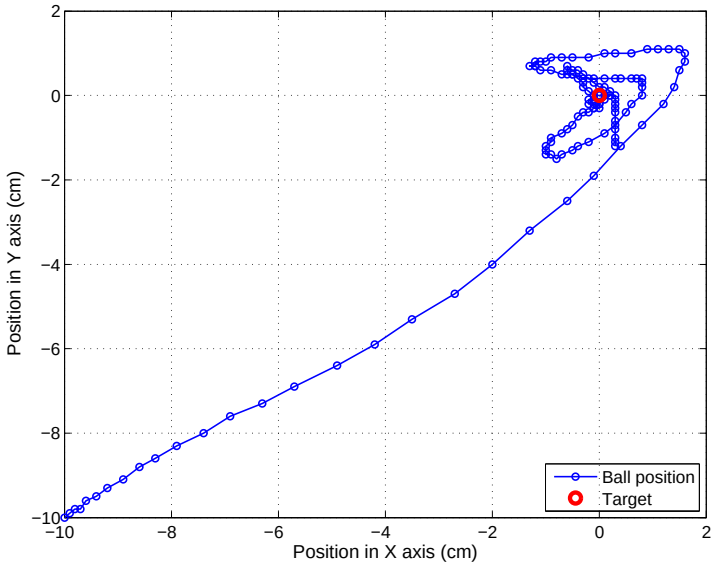


Figure 6.10: Double axis control with a different starting position.

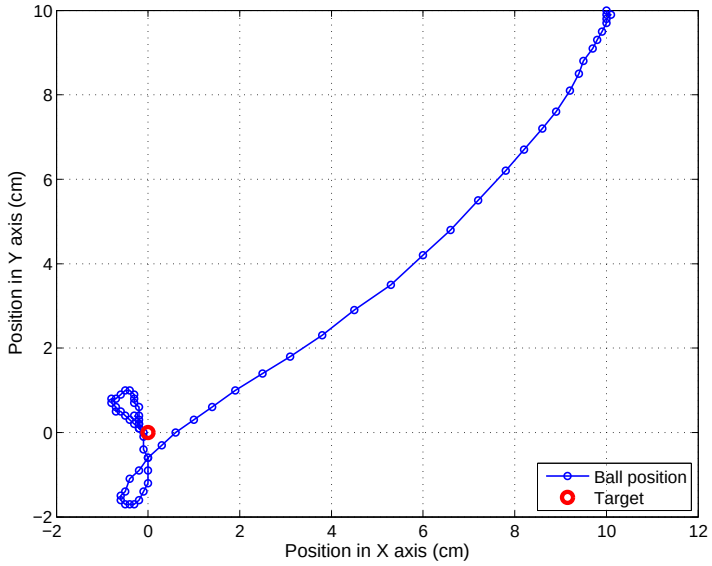


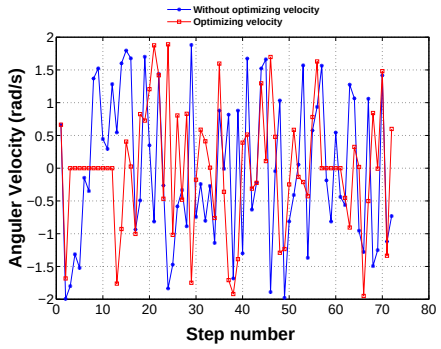
Figure 6.11: Double axis control with modified cost function.

the cost function. The cost function can be modified as

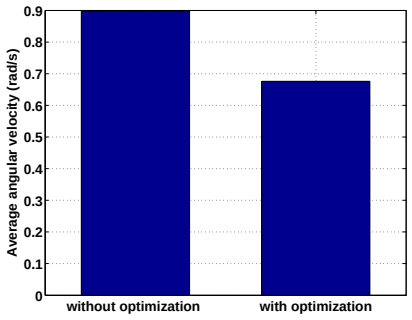
$$\mathbf{J}(\mathbf{x}_k, \mathbf{u}_k) = (x_t - x_k)^2 + (x_t - x_k)^2 + (u1_k)^2 + (u2_k)^2 \tag{6.6}$$

Figure 6.11 shows the trajectory generated using the modified cost function. Figure 6.12 shows the control velocities for joints $a5$ and $a6$. Figures 6.12(a) and 6.12(c) show the applied velocity at each time step for joints $a5$ and $a6$ respectively. Similarly, in Figures 6.12(b) and 6.12(d), the average velocity applied in joint $a5$ and $a6$ is shown. From Figure 6.12, it is evident that including joint velocity in the cost function, it is possible to generate an optimized trajectory, with a lower range of control velocity.

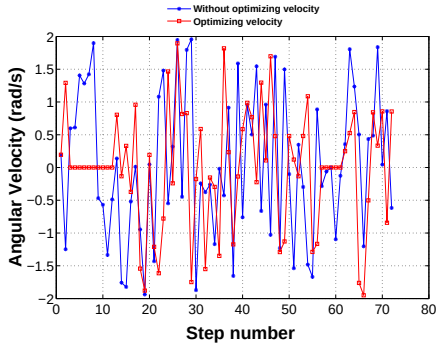
From this test, we can conclude that the designed controller is capable of handling systems with two variables. However, a noticeable aspect of the experiment results was that the ball did not become completely steady at the final position, instead, it moved within a range of ±1cm of the target position in both x and y-axis.



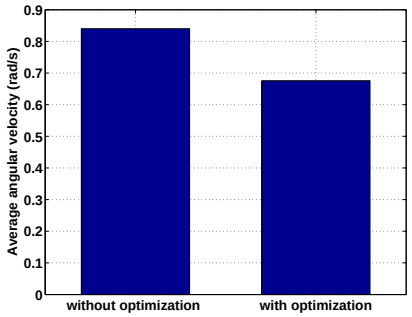
(a) Applied control velocity to joint $a5$.



(b) Average velocity in joint $a5$.



(c) Applied control velocity to joint $a6$.



(d) Average velocity in joint $a6$.

Figure 6.12: Applied control velocity and the average velocity of joints $a5$ and $a6$.

6.4 Robot tray balancing: Multivariable control with constraints

The objective of this test was to evaluate the ability of the controller to handle a multivariable system with constraints. In this experiment, an obstacle was placed between the starting and the target position to represent workspace constraints. For this experiment also the same simulation parameters in Table 6.1 and optimization parameters in Table 6.2 were used.

For the designed controller, the number of trajectories generated at each prediction phase plays an important role for a multivariable system with constraints. It is difficult to find an optimized trajectory for a system with a higher number of variables and constraints if the number of samples is too small. On the other hand, use of a higher number of samples increases the simulation and prediction time. Similarly, the length of the prediction horizon also has an important

effect, especially when constraints are present in the system.

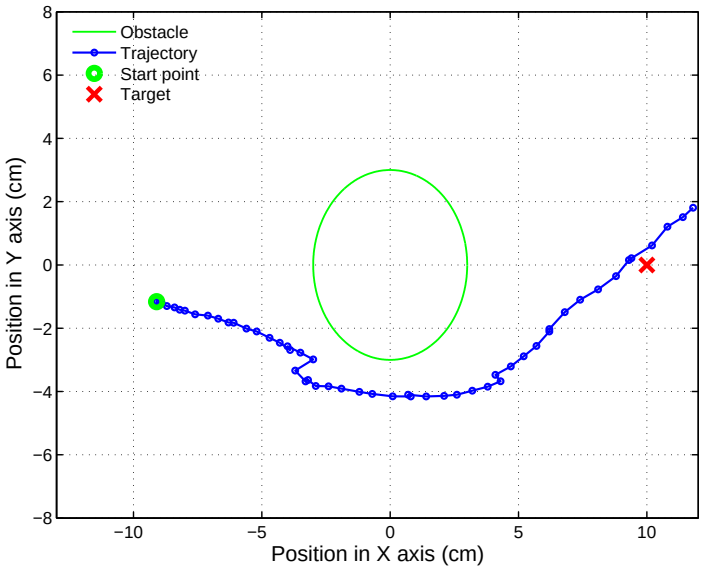
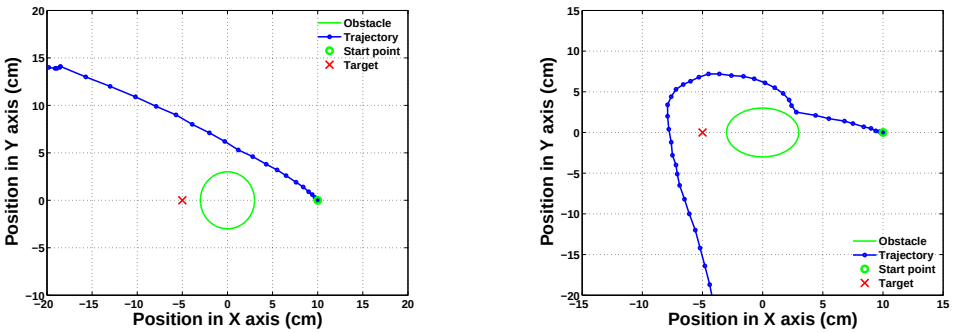


Figure 6.13: Trajectory with an obstacle between starting and target position for the real sytem.

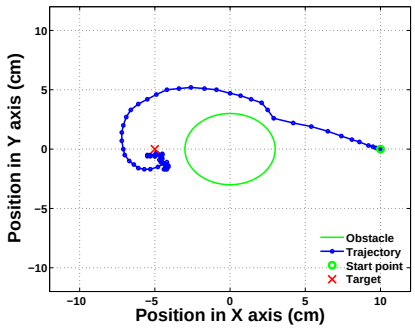
Figure 6.13 shows the trajectory of the controlled ball for the real system with an obstacle between the starting and the target position. In this case, sample number N = 50 and prediction horizon of $N_p = 15$ time step were used. It is evident from the Figure 6.13 that even though the controller was successful in avoiding the obstacle, it was unable to balance the ball at the target position. To ensure the validity of the result total 30 trials were carried out with the same parameters, among which Figure 6.13 was the best result.

To choose the best optimization parameters for the current task, a number of test were carried out in the simulator, with different sample numbers and prediction horizons. Figure 6.14 shows the trajectories for different prediction horizons using 50 samples in the simulator. From the Figure 6.14 it is evident that for smaller prediction horizons ($N_p = 10$ and 15 time step), the controller was unable to complete the task in the simulator. However, for $N_p = 20$ time step the controller completed the task with 50 samples in the simulator.

Figure 6.15 shows trajectories for different prediction horizons, for sample number $N = 60$ in the simulator. From this figure, it is apparent that with 60 samples the controller was able to complete the task for all the perdition horizon values ($N_p = 10$, 15, 20 time step). However, for the small prediction horizon N_p=10 time step, the controller did not provide a good trajectory.



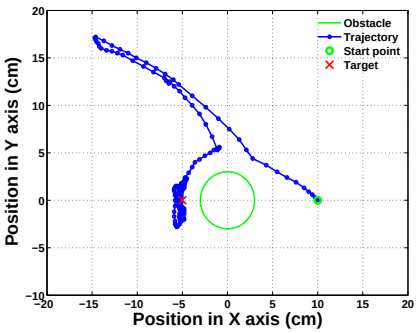
(a) Prediction horizon $N_p = 10$ time step. (b) Prediction horizon $N_p = 15$ time step.



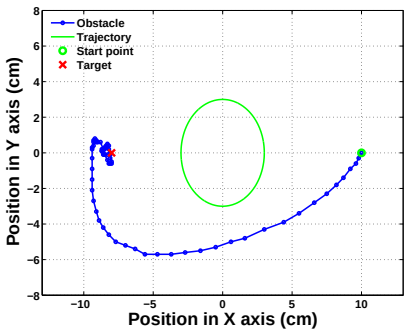
(c) Prediction horizon $N_p = 20$ time step.

Figure 6.14: Trajectory for different prediction horizons in the simulator, with sample number N =50. Markers in the trajectory represent the time step.

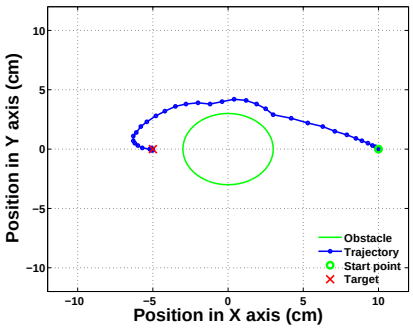
From the results in Figures 6.14 and 6.15, we can say that for the current task the combinations $N_p = 20$, N= 50 and $N_p = 15$, N= 60 are the best possible combinations for which it was possible to complete the task in the simulator. Moreover, to check the consistency of the result, test was carried out for different starting and ending position using both the combinations, which is shown in Figure 6.16.



(a) Prediction horizon $N_p = 10$ time step.

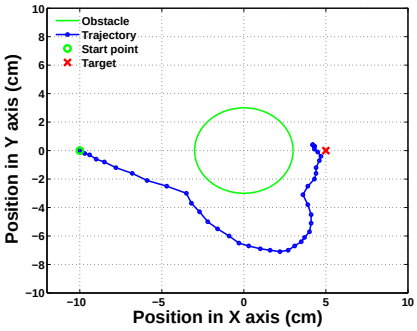


(b) Prediction horizon $N_p = 15$ time step.

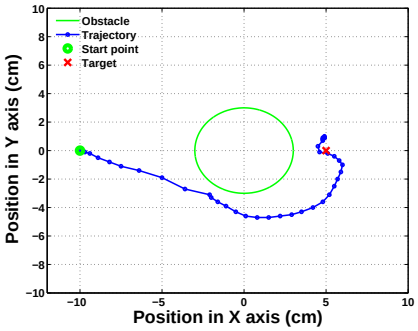


(c) Prediction horizon $N_p = 20$ time step.

Figure 6.15: Trajectory for different prediction horizons in simulator, with sample number N =60, Markers in the trajectory represent the time step.



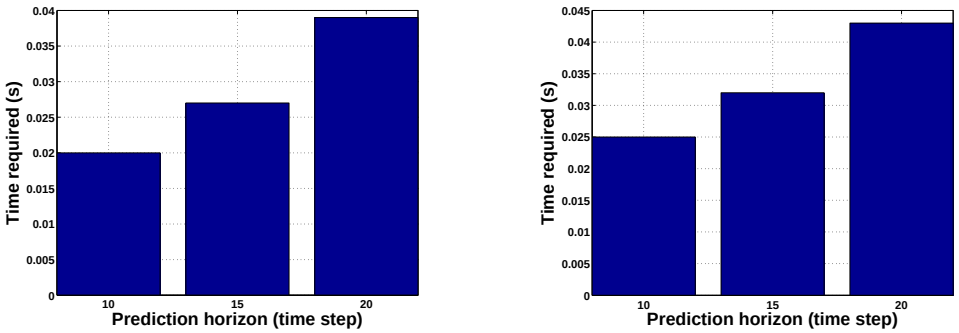
(a) $N_p = 20$, N= 50.



(b) $N_p = 15$, N= 60.

Figure 6.16: Trajectory for different starting and target point in simulator.

Increase in either sample number or prediction horizon increases the required simulation time. Figure 6.17 shows the required time for simulation and optimization for different sample size and prediction horizon.



(a) For sample number N= 50.

(b) For sample number N= 60.

Figure 6.17: Time required for simulation and optimization.

From the Figure 6.17, it is evident that increase in prediction horizon or sample number increases the optimization time. In this case, for both the combination chosen from simulation, the controller was not capable of operating in real-time.

6.5 Discussion

A typical model predictive controller is superior to other conventional controllers due to its ability to handle nonlinear systems with multiple variables and constraint. The purpose of the experiments was to evaluate above mentioned functionality of the designed model predictive controller.

The objective of the first experiment (Section 6.1) was to check the quality of the simulation for the built system. The results of the experiment suggest that using ODE it was difficult to achieve highly accurate simulation. One of the main reasons behind the discrepancy between the simulation and real world was that the ODE physics engine emphasizes more on the performance of the simulation rather than the accuracy, but the accuracy of the simulation can be increased by tuning the simulation parameters. However, tuning the simulation parameters is application specific and does not provide a general solution for increasing the accuracy of the simulation.

From the result of the second and third experiment (Section 6.2 ad 6.3), it is evident that the system is capable of controlling a simple nonlinear system for both single variable and multiple variables. However, for the multivariable system only two variables were considered. The capability of the controller to handle a higher number of variables was not explored in this work.

The aim of the fourth experiment (Section 6.4) was to the test the constraint handling capability of the controller for a multivariable nonlinear system. The

result of the experiment shows that the controller was unable to complete the task for a constrained multivariable system in real-time. The main reason behind the failure of the controller was the simulation speed of the ODE physics engine. From the experiment, it was evident that to successfully complete the task a higher sample number or a longer prediction horizon was required. Unfortunately, using a higher number of samples or longer prediction horizon increases the simulation time for which real-time optimization was not possible.

Considering the overall results of the thesis work, the main difficulties behind an implementation of physics engine based model predictive controller were the accuracy and the speed of the simulation. Implemented controller corrects the error between the simulation and the real world at each time step, which essentially reduces the effect of inaccuracy of the simulation. However, simulation speed of the ODE physics engine still remained as the major difficulty, especially when the complexity of the system increases. Nevertheless, the simulation time can be reduced by using a faster computer and using a better computation technique. For example, in this work, the prediction of the states was performed in serial manner using only one processor at a time. The speed of the predictions can be increased by using parallel computation techniques by ensuring the simultaneous use of all four available processors. However, using a faster computer may solve the problem for this particular system and the task, but to solve the problem for general use with more complex systems, a faster simulator is required.

Another important setback for this work was the speed of the camera based feedback system. Due to the slow nature of processing speed of the camera, a larger simulation step size needed to be chosen, which had a significant effect on the accuracy of the simulation. However, this problem can be resolved by using a faster feedback approach.

Irrespective of the difficulties discussed, the proposed approach showed promise as all the faced problems can solved with further work. In this approach, the design process of the controller is much simpler than other conventional techniques of model predictive controls and provides opportunity to use it for a range of robotic systems. Moreover, introduction of constraints to the system is easier than the conventional mathematical formulation, for example, in the fourth experiment (section 6.4), a workspace constraint was imposed by simply adding an obstacle between starting and target point into the simulation model, which is not trivial when a mathematical model is used. Moreover, constraints on the control inputs can also be imposed by limiting the range of values for the generated input samples. Similarly, with this approach cost functions can be formed and modified more intuitively . An example of the effect of cost function modification was shown in Figure 6.12, which suggests that the same task can be completed by applying a lower range of velocities.

Chapter 7

Conclusion

The objective of the thesis work was to provide a generalized solution for implementing model predictive control for robotic applications. In this approach, a physics engine was used for prediction of the states and a stochastic optimization technique was used for optimization.

One of the most important aspects of the thesis work was to ensure accurate simulation. Physics engine ODE was chosen among the available open source physics engines. The reason behind the choice of ODE was its accuracy compared to other available physics engines. Even though ODE is not the fastest of the available physics engines, it provides a balanced performance considering the speed and accuracy of the simulation.

A significant part of the thesis was to design a proper optimization technique for a real robotic system. As the aim of the thesis was to implement the MPC algorithm for a nonlinear system, a stochastic optimization based on particle belief propagation technique was chosen. To test the functionality of the controller, a nonlinear robotic system was built and to control the robotic system a software system was built. The quality of the simulation for the built system was tested and appropriate simulation parameters were chosen for the current system.

Finally to test the functionality of the designed controller, different experiments were designed. The objective of the tests was to evaluate the performance of the controller for a nonlinear system. From the results of the tests, it can be concluded that the designed controller is capable of handling a nonlinear system with single and multiple variables. However, for a nonlinear multivariable system with constraints the controller is unable to operate in real time.

The proposed approach is potentially usable for a range of robotic applications depending on the capability of the physics engine used. Even though, in this

case, the controller was unable to handle constraints for a multivariable system, from the results it is clear that with a faster simulator the problem is solvable.

The idea of using a physics engine for prediction of states in MPC context is relatively new. However, the approach used in [24] is similar to this work in a sense that a physics engine was used for implementing MPC. The work was done in simulation and any implementation on a real robotic system is yet to be reported. The most relevant work to this thesis can be found in [18], where the MPC controller was designed for animation purpose. In this approach also a physics engine was used for state prediction and an optimization technique based on Particle belief propagation was used. In contrast, while both the works were limited to simulations, in this thesis an implementation in a real robotic system is presented.

Currently, available physics engines are limited in their capabilities for robotic applications. Although, ODE was found to be the best available simulator for robotic applications from several comparative studies between available physics engines, it is still not sufficient for simulation-based controller design for complex systems regarding its speed and accuracy. Even though the robotic hardware is getting increasingly complex every day, simulation tools for the robotic systems have not improved proportionally. Most of the currently available open source physics engines were designed for animation, where the motivation was to provide visually-plausible simulation rather than physically accurate simulation. A fast and accurate simulator explicitly for robotic system is still unavailable which makes it difficult to implement any simulation-based approach for controller design. With an advanced physics engine, it will be possible not only to implement model-based controllers but also design more sophisticated controllers for complex robotic systems using other control techniques.

References

[1] L. Zlajpah, "Simulation in robotics," *Mathematics and Computers in Simulation*, vol. 79, no. 4, pp. 879–897, 2008.

[2] J. Nocedal and S. J. Wright, *Numerical Optimization*, ser. Springer Series in Operations Research and Financial Engineering. London, United Kingdom: Springer New York, 1999.

[3] M. C. Fu, "Optimization for simulation: Theory vs. practice," *INFORMS Journal on Computing*, vol. 14, no. 3, pp. 192–215, 2002.

[4] E. Camacho and C. Bordons, *Model Predictive Control*, 2nd ed., ser. Advanced Textbooks in Control and Signal Processing. London, United Kingdom: Springer-Verlag London, 2007.

[5] J. Spall, "Stochastic optimization," in *Handbook of Computational Statistics*, ser. Springer Handbooks of Computational Statistics. Springer Berlin Heidelberg, 2012, pp. 173–201.

[6] A. Ihler and D. McAllester, "Particle belief propagation," *Journal of Machine Learning Research*, vol. 5, pp. 256–263, 2009.

[7] C. E. García, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335 – 348, 1989.

[8] A. Ollero and O. Amidi, "Predictive path tracking of mobile robots. Application to the CMU Navlab," in *International Conference on Advanced Robotics*, vol. 91, 1991, pp. 1081–1086.

[9] J. E. Normey-Rico, J. G. Ortega, and E. F. Camacho, "A smith-predictor-based generalised predictive controller for mobile robot path-tracking," *Control Engineering Practice*, vol. 7, no. 6, pp. 729 – 740, 1999.

[10] F. Künhe, W. F. Lages, and J. M. G. da Silva Jr, "Model predictive control of a mobile robot using linearization," in *Proceedings of Mechatronics and Robotics*, 2004, pp. 525–530.

[11] F. Künhe, W. F. Lages, and J. M. G. da Silva Jr, "Mobile robot trajectory tracking using model predictive control," in *II IEEE latin-american robotics symposium*, 2005.

[12] P. Boscariol, A. Gasparetto, and V. Zanotto, "Model predictive control of a flexible links mechanism," *Journal of Intelligent and Robotic Systems*, vol. 58, no. 2, pp. 125–147, 2010.

[13] J. G. Ortega and E. F. Camacho, "Mobile robot navigation in a partially structured static environment, using neural predictive control," *Control Engineering Practice*, vol. 4, no. 12, pp. 1669 – 1679, 1996.

[14] B. Song and A. J. Koivo, "Nonlinear predictive control with application to manipulator with flexible forearm," *IEEE Transactions on Industrial Electronics*, vol. 46, no. 5, pp. 923–932, Oct 1999.

[15] J. Zhao, M. Diehl, R. Longman, H. Bock, and J. Schlöder, "Nonlinear model predictive control of robots using real-time optimization," in *Collection of Technical Papers - AIAA/AAS Astrodynamics Specialist Conference*, vol. 2, 2004, pp. 1145–1162.

[16] M. Fu, S. Andradottir, J. Carson, F. Glover, C. Harrell, Y.-C. Ho, J. Kelly, and S. Robinson, "Integrating optimization and simulation: research and practice," in *Winter Simulation Conference*, vol. 1, 2000, pp. 610–616.

[17] T. Erez, Y. Tassa, and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," in *IEEE International Conference on Robotics and Automation*, May 2015, pp. 4397–4404.

[18] P. Hämäläinen, J. Rajamäki, and C. K. Liu, "Online control of simulated humanoids using particle belief propagation," in *International Conference and Exhibition on Computer Graphics and Interactive Techniques*, New York, NY, USA, 2015.

[19] M. Vidyasagar, "Randomized algorithms for robust controller synthesis using statistical learning theory," *Automatica*, vol. 37, no. 10, pp. 1515 – 1528, 2001.

[20] T. Roberto, T. Roberto, C. Giuseppe, and F. Dabbene, *Randomized Algorithms for Analysis and Control of Uncertain Systems*, 2nd ed. London, United Kingdom: Springer-Verlag London, 2013.

[21] J. Piovesan and H. Tanner, "Randomized model predictive control for robot navigation," in *IEEE International Conference on Robotics and Automation*, May 2009, pp. 94–99.

[22] H. Tanner and J. Piovesan, "Randomized receding horizon navigation," *IEEE Transactions on Automatic Control*, vol. 55, no. 11, pp. 2640–2644, Nov 2010.

[23] X. Zhang, K. Margellos, P. Goulart, and J. Lygeros, "Stochastic model predictive control using a combination of randomized and robust optimization," in *IEEE Annual Conference on Decision and Control*, Dec 2013, pp. 7740–7745.

[24] T. Erez, K. Lowrey, Y. Tassa, V. Kumar, S. Kolev, and E. Todorov, "An integrated system for real-time model predictive control of humanoid robots," in *IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, October 2013, pp. 292–299.

[25] V. Kumar, Y. Tassa, T. Erez, and E. Todorov, "Real-time behaviour synthesis for dynamic hand-manipulation," in *IEEE International Conference on Robotics and Automation*, May 2014, pp. 6808–6815.

[26] P. Hämäläinen, S. Eriksson, E. Tanskanen, V. Kyrki, and J. Lehtinen, "Online motion synthesis using sequential monte carlo," *ACM Transactions on Computer Systems*, vol. 33, no. 4, 2014.

[27] A. Bemporad, M. Morari, V. Dua, and E. N. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3 – 20, 2002.

[28] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733 – 764, 2003.

[29] A. Bemporad and M. Morari, "Robust Model Predictive Control: A Survey," *Robustness in identification and control SE - 16*, vol. 245, pp. 207–226, 1999.

[30] M. Cannon, "Efficient nonlinear model predictive control algorithms," *Annual Reviews in Control*, vol. 28, no. 2, pp. 229 – 237, 2004.

[31] D. Clarke, C. Mohtadi, and P. Tuffs, "Generalized predictive control—Part I. The basic algorithm," *Automatica*, vol. 23, pp. 137–148, 1987.

[32] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, vol. 23, pp. 667–682, 1999.

[33] E. Camacho and C. Bordons, "Nonlinear model predictive control: An introductory review," in *Assessment and Future Directions of Nonlinear Model Predictive Control*, ser. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2007, vol. 358, pp. 1–16.

[34] M. A. Henson, "Nonlinear model predictive control: current status and future directions," *Computers & Chemical Engineering*, vol. 23, no. 2, pp. 187 – 202, 1998.

[35] D. Gu and H. Hu, "Neural predictive control for a car-like mobile robot," *Robotics and Autonomous Systems*, vol. 39, no. 2, pp. 73 – 86, 2002.

[36] S. J. Yoo, Y. H. Choi, and J. B. Park, "Generalized predictive control based on self-recurrent wavelet neural network for stable path tracking of mobile robots: adaptive learning rates approach," *IEEE Transactions on Circuits and Systems I: Regular Papers,*, vol. 53, no. 6, pp. 1381–1394, June 2006.

[37] Y. Yang, X. Lin, Z. Miao, X. Yuan, and Y. Wang, "Predictive control strategy based on extreme learning machine for path-tracking of autonomous mobile robot," *Intelligent Automation & Soft Computing*, vol. 21, no. 1, pp. 1–19, 2015.

[38] Z. Wei and G. Fang, "Model Predictive Control for Robot Manipulators Using A Neural Network Model," *Australasian Conference on Robotics and Automation*, pp. 62–67, 1999.

[39] J. Zhao, M. Diehl, and R. Longman, "Nonlinear Model Predictive Control of Robots Using Real-Time Optimization," in *Astrodynamic Specialist Conference and Exhibit*, no. August, Providance, Rhode Island, August 2004, pp. 1–18.

[40] B. Durmuş, H. Temurtaş, N. Yumuşak, and F. Temurtaş, "A study on industrial robotic manipulator model using model based predictive controls," *Journal of Intelligent Manufacturing*, vol. 20, no. 2, pp. 233–241, 2009.

[41] D. Dimitrov, P. Wieber, H. Ferreau, and M. Diehl, "On the implementation of model predictive control for on-line walking pattern generation," in *IEEE International Conference on Robotics and Automation*, May 2008, pp. 2685–2690.

[42] H. Diedam, D. Dimitrov, P. Wieber, K. Mombaur, and M. Diehl, "Online walking gait generation with adaptive foot positioning through linear model predictive control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept 2008, pp. 1121–1126.

[43] C. Azevedo, P. Poignet, and B. Espiau, "Moving horizon control for biped robots without reference trajectory," in *IEEE International Conference on Robotics and Automation*, vol. 3, 2002, pp. 2762–2767.

[44] Z. Zhu, Y. Wang, and X. Chen, "Real-Time Control of Full Actuated Biped Robot Based on Nonlinear Model Predictive Control," *Lecture Notes in Computer Science*, vol. 5314, pp. 873–882.

[45] B. Henze, C. Ott, and M. Roa, "Posture and balance control for humanoid robots in multi-contact scenarios based on model predictive control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2014, pp. 3253–3258.

[46] S. Piperakis, E. Orfanoudakis, and M. Lagoudakis, "Predictive control for dynamic locomotion of real humanoid robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, September 2014, pp. 4036–4043.

[47] H. Kim, D. Shim, and S. Sastry, "Nonlinear model predictive tracking control for rotorcraft-based unmanned aerial vehicles," in *American Control Conference*, vol. 5, 2002, pp. 3576–3581 vol.5.

[48] D. Shim, H. Kim, and S. Sastry, "Decentralized nonlinear model predictive control of multiple flying robots," in *IEEE Conference on Decision and Control*, vol. 4, December 2003, pp. 3621–3626 vol.4.

[49] T. Keviczky and G. J. Balas, "Receding horizon control of an f-16 aircraft: A comparative study," *Control Engineering Practice*, vol. 14, no. 9, pp. 1023 – 1033, 2006.

[50] R. Ginhoux, J. Gangloff, M. de Mathelin, L. Soler, M. Sanchez, and J. Marescaux, "Active filtering of physiological motion in robotized surgery using predictive control," *IEEE Transactions on Robotics*, vol. 21, no. 1, pp. 67–79, Feb 2005.

[51] O. Bebek and M. Cenk, "Intelligent Control Algorithms for Robotic-Assisted Beating Heart Surgery," *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 468–480, 2007.

[52] M. Dominici and R. Cortesao, "Model predictive control architectures with force feedback for robotic-assisted beating heart surgery," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 2276–2282.

[53] A. Ihler, J. Fisher III, and A. Willsky, "Loopy belief propagation: Convergence and effects of message errors," *Journal of Machine Learning Research*, vol. 6, pp. 905–936, 2005.

[54] D. Koller, N. Friedman, L. Getoor, and B. Taskar, "Graphical models in a nutshell," in *Introduction to statistical relational learning*, Cambridge:MIT Press, 2007, pp. 13-55.

[55] C. Wang, N. Komodakis, and N. Paragios, "Markov random field modeling, inference and learning in computer vision and image understanding: A survey," *Computer Vision and Image Understanding*, vol. 117, no. 11, pp. 1610–1627, 2013.

[56] B. Taskar, V. Chatalbashev, and D. Koller, "Learning associative markov networks," in *International Conference on Machine Learning*, Banff, Alberta, Canada, 2004, pp. 807–814.

[57] J. Yedidia, W. Freeman, and Y. Weiss, "Generalized belief propagation," in *Advances in Neural Information Processing Systems*, Denver, United States, 2001.

[58] D. Gamarnik, D. Shah, and Y. Wei, "Belief propagation for min-cost network flow: Convergence and correctness," in *Annual ACM-SIAM Symposium on Discrete Algorithms*, Austin, TX; United States,, Austin, TX; United States, 2010, pp.279-292, pp. 279–292.

[59] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*. San Mateo: Morgan Kaufman, 1988.

[60] R. Kothapa, J. Pacheco, and E. Sudderth, "Max-product particle belief propagation," Brown University Dept. of Computer Science, Tech. Rep., November 2001.

[61] A. R. Mustafa, "Parallelization of the webots simulation loop and the ode physics engine," Master's thesis, Biorobotics Laboratory, École Polytechnique Fédérale de Lausanne, Switzerland, 2014.

[62] K. Erleben, "Module based design for rigid body simulators," Department of Computer Science,University of Copenhagen, Copenhagen,Denmark, Tech. Rep. DIKU-TR-02/06, July 2012.

[63] Open Dynamic Engine, "Products that use ode," (accessed July 27, 2015). [Online]. Available: http://ode-wiki.org/wiki/index.php?title=Products_that_use_ODE"

[64] Bulletphysics.org, "Real-time physics simulation," (accessed July 27, 2015). [Online]. Available: http://bulletphysics.org/wordpress/

[65] J. Hummel, R. Wolff, T. Stein, A. Gerndt, and T. Kuhlen, "An evaluation of open source physics engines for use in virtual reality assembly simulations," in *Advances in Visual Computing*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7432, pp. 346–357.

[66] A. Roennau, F. Sutter, G. Heppner, J. Oberlaender, and R. Dillmann, "Evaluation of physics engines for robotic simulations with a special focus on the dynamics of walking robots," in *International Conference on Advanced Robotics*, November 2013, pp. 1–7.

[67] K. Kumar and P. S. Reel, "Analysis of contemporary robotics simulators," in *International Conference on Emerging Trends in Electrical and Computer Technology*, March 2011, pp. 661–665.

[68] M. Torres-Torriti, T. Arredondo, and P. Castillo-Pizarro, "Survey and comparative study of free simulation software for mobile robots," *Robotica*, pp. 1–32, 7 2015.

[69] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura; United States, 2012, pp. 5026–5033.

[70] E. Drumwright, J. Hsu, N. Koenig, and D. Shell, "Extending open dynamics engine for robotics simulation," in *Simulation, Modeling, and Programming for Autonomous Robots*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, vol. 6472, pp. 38–50.

[71] R. Smith, "Open dynamic engine, v0.5 user guide," (accessed July 27, 2015). [Online]. Available: http://ode-wiki.org/wiki/index.php?title=Manual

[72] E. Todorov, "General duality between optimal control and estimation," in *IEEE Conference on Decision and Control*, December 2008, pp. 4286–4292.

[73] F. Gustafsson, "Particle filter theory and practice with positioning applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 25, no. 7, pp. 53–82, July 2010.

[74] G. Schreiber, A. Stemmer, and R. Bischoff, "The fast research interface for the kuka lightweight robot," in *IEEE Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications*, 2010.

[75] F. Steinmetz, "Programming by demonstration for incontact tasks using dynamic movement primitives," Master's thesis, Luleå University of Technology, Kiruna, Sweden, 2014.

[76] The Orocos Project, "Smarter control in robotics & automation!" (accessed July 27, 2015). [Online]. Available: http://www.test.org/doe/

[77] ros.org, "Robot operating system," (accessed July 27, 2015). [Online]. Available: http://www.ros.org/

[78] Openni.ru, "Open-source SDK for 3d sensors," (accessed July 27, 2015). [Online]. Available: http://openni.ru/

[79] OpenCV.org, "Open source computer vision," (accessed July 27, 2015). [Online]. Available: http://opencv.org/

[80] Microsoft, "Kinect for windows sensor components and specifications," (accessed July 27, 2015). [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131033.aspx

[81] J. Hsu and S. Peters, "Extending open dynamics engine for the darpa virtual robotics challenge," in *Simulation, Modeling, and Programming for Autonomous Robots*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2014, vol. 8810, pp. 37–48.